

WebFOCUS

WebFOCUS Maintain Getting Started
Version 5 Release 2

EDA, EDA/SQL, FIDEL, FOCCALC, FOCUS, FOCUS Fusion, FOCUS Vision, Hospital-Trac, Information Builders, the Information Builders logo, Parlay, PC/FOCUS, SmartMart, SmartMode, SNAPpack, TableTalk, WALDO, Web390, WebFOCUS and WorldMART are registered trademarks, and iWay and iWay Software are trademarks of Information Builders, Inc.

Due to the nature of this material, this document refers to numerous hardware and software products by their trademarks. In most, if not all cases, these designations are claimed as trademarks or registered trademarks by their respective companies. It is not this publisher's intent to use any of these names generically. The reader is therefore cautioned to investigate all claimed trademark rights before using any of these names other than to refer to the product described.

Copyright © 2003, by Information Builders, Inc. All rights reserved. This manual, or parts thereof, may not be reproduced in any form without the written permission of Information Builders, Inc.

Printed in the U.S.A.

Preface

This documentation provides an introduction to WebFOCUS Maintain. It is intended for application developers who are responsible for planning the enterprise's software environment. This documentation is part of the WebFOCUS Maintain documentation set.

How This Manual Is Organized

This manual includes the following chapters:

Chapter/Appendix		Contents
1	Introducing WebFOCUS Maintain	Familiarizes you with the architecture and features of WebFOCUS Maintain. It also contains a step-by-step overview of developing and deploying applications.
2	The WebFOCUS Maintain Basic Tutorial	Leads you, step-by-step, through the process of creating and running a WebFOCUS Maintain application.
3	The WebFOCUS Maintain Advanced Tutorial	Continues with developing the application introduced in the Basic Tutorial, introducing more advanced topics.
4	WebFOCUS Maintain Concepts	Contains an overview of WebFOCUS Maintain concepts, including set-based processing, controlling an application's flow, forms and event-driven processing, reading and writing records, transaction processing, and object-oriented application development.

About Your Documentation

WebFOCUS Maintain documentation includes one printed manual, a context-sensitive help system, and three corresponding manuals that are provided as PDF files. The manuals provided are:

- *Getting Started*, which is this manual (print and PDF).
- *Developing WebFOCUS Maintain Applications* describes how to develop and deploy WebFOCUS Maintain applications using the Maintain Development Environment (PDF only).

- *Language Reference* describes the Maintain language, including basic language rules, a comprehensive reference of Maintain commands and expressions, form and control properties, and error messages. (PDF only.)

Since WebFOCUS Maintain is part of the WebFOCUS Developer Studio suite of tools, you will also need to have available your WebFOCUS Developer Studio documentation.

Documentation Conventions

The following conventions apply throughout this manual:

Convention	Description
THIS TYPEFACE or <i>this typeface</i>	Denotes syntax that you must enter exactly as shown.
<i>this typeface</i>	Represents a placeholder (or variable) in syntax for a value that you or the system must supply.
<u>underscore</u>	Indicates a default setting.
<i>this typeface</i>	Represents a placeholder (or variable) in a text paragraph, a cross-reference, or an important term. It may also indicate a button, menu item, or dialog box option you can click or select.
this typeface	Highlights a file name or command in a text paragraph that must be lowercase.
Key + Key	Indicates keys that you must press simultaneously.
{ }	Indicates two or three choices; type one of them, not the braces.
[]	Indicates a group of optional parameters. None are required, but you may select one of them. Type only the parameter in the brackets, not the brackets.
	Separates mutually exclusive choices in syntax. Type one of them, not the symbol.
...	Indicates that you can enter a parameter multiple times. Type only the parameter, not the ellipsis points (...).
.	Indicates that there are (or could be) intervening or additional commands.

Related Publications

The WebFOCUS Developer Studio documentation set may also contain information that is useful to you.

Visit our World Wide Web site, <http://www.informationbuilders.com>, to view a current listing of our publications and to place an order. You can also contact the Publications Order Department at (800) 969-4636.

Customer Support

Do you have questions about WebFOCUS Maintain?

Call Information Builders Customer Support Service (CSS) at (800) 736-6130 or (212) 736-6130. Customer Support Consultants are available Monday through Friday between 8:00 a.m. and 8:00 p.m. EST to address all your WebFOCUS Maintain questions. Information Builders consultants can also give you general guidance regarding product capabilities and documentation. Please be ready to provide your six-digit site code (xxxx.xx) when you call.

You can also access support services electronically, 24 hours a day, with InfoResponse Online. InfoResponse Online is accessible through our World Wide Web site, <http://www.informationbuilders.com>. It connects you to the tracking system and known-problem database at the Information Builders support center. Registered users can open, update, and view the status of cases in the tracking system and read descriptions of reported software issues. New users can register immediately for this service. The technical support section of www.informationbuilders.com also provides usage techniques, diagnostic tips, and answers to frequently asked questions.

To learn about the full range of available support services, ask your Information Builders representative about InfoResponse Online, or call (800) 969-INFO.

Information You Should Have

To help our consultants answer your questions most effectively, be ready to provide the following information when you call:

- Your six-digit site code (xxxx.xx).
- Your WebFOCUS configuration:
 - The front-end you are using, including vendor and release.
 - The communications protocol (for example, TCP/IP or HLLAPI), including vendor and release.
 - The software release.
 - The server you are accessing, including release (for example, 5.2).

- The stored procedure (preferably with line numbers) or FOCUS commands being used in server access.
- The name of the Master File and Access File.
- The exact nature of the problem:
 - Are the results or the format incorrect? Are the text or calculations missing or misplaced?
 - The error message and return code, if applicable.
 - Is this related to any other problem?
- Has the procedure or query ever worked in its present form? Has it been changed recently? How often does the problem occur?
- What release of the operating system are you using? Has it, WebFOCUS, your security system, communications protocol, or front-end software changed?
- Is this problem reproducible? If so, how?
- Have you tried to reproduce your problem in the simplest form possible? For example, if you are having problems joining two data sources, have you tried executing a query containing the code to access a single data source?
- Do you have a trace file?
- How is the problem affecting your business? Is it halting development or production? Do you just have questions about functionality or documentation?

User Feedback

In an effort to produce effective documentation, the Documentation Services staff welcomes any opinion you can offer regarding this manual. Please use the Reader Comments form at the end of this manual to relay suggestions for improving the publication or to alert us to corrections. You can also use the Documentation Feedback form on our Web site, <http://www.informationbuilders.com>.

Thank you, in advance, for your comments.

Information Builders Consulting and Training

Interested in training? Information Builders Education Department offers a wide variety of training courses for this and other Information Builders products.

For information on course descriptions, locations, and dates, or to register for classes, visit our World Wide Web site (<http://www.informationbuilders.com>) or call (800) 969-INFO to speak to an Education Representative.

Contents

1. Introducing WebFOCUS Maintain	1-1
Road Map: Where Should You Go?	1-2
What Is WebFOCUS Maintain? (for New Users)	1-2
The Challenge of Accessing Information	1-2
How WebFOCUS Maintain Works	1-2
N-Tier Applications	1-3
Leveraging the Power of the WebFOCUS Server	1-3
In Summary: What WebFOCUS Maintain Can Do for You	1-4
What's Next?	1-4
Overview of Developing WebFOCUS Maintain Applications	1-4
2. The WebFOCUS Maintain Basic Tutorial	2-1
Before You Begin	2-2
Creating a Project	2-2
Familiarizing Yourself With the Explorer	2-6
Adding a Data Source to Your Project	2-7
Familiarizing Yourself With the Project Explorer	2-8
More About the Project Explorer: Viewing the Structure of a Data Source	2-9
Designing a Form	2-11
What Are Data Source Stacks?	2-12
More About the Project Explorer: More About the Components of a Procedure	2-14
Working in the Form Editor	2-14
Moving Controls on a Form	2-15
Renaming Field Prompts	2-16
Aligning Controls	2-18
Saving Your Work	2-20
Running Your Project Locally	2-21
Using Radio Buttons	2-22
Stacks and Implied Columns	2-26
Giving Your Form a New Title	2-26
Writing Data to the Data Source	2-27
Writing Functions	2-28
About the Procedure Editor	2-29
Clearing Data From Stacks	2-34
Assigning the Function to the Add Button	2-35
Adding a Form to Display Data From a Data Source	2-37
Extracting Data From a Data Source Into a Stack	2-38
Adding an HTML Table to Your Form	2-39
Creating a Link From One Form to Another	2-42
Using the HTML Overflow Property	2-44

Adding Form Navigation Buttons	2-44
Adding Images to Your Project	2-46
Tiling Images	2-50
Using Folders in the Explorer	2-50
What's Next?	2-53
3. The WebFOCUS Maintain Advanced Tutorial	3-1
4. WebFOCUS Maintain Concepts	4-1
Set-based Processing	4-2
Which Processes Are Set-based?	4-2
How Does Maintain Process Data in Sets?	4-4
Creating and Defining Data Source Stacks: An Overview	4-4
Creating a Data Source Stack	4-5
Defining a Data Source Stack's Data Source Columns	4-5
Creating a Data Source Stack's User-defined Columns	4-8
Copying Data Into and Out of a Data Source Stack	4-9
Referring to Specific Stack Rows Using an Index	4-10
Looping Through a Stack	4-11
Sorting a Stack	4-12
Editing and Viewing Stack Values	4-12
The Default Data Source Stack: The Current Area	4-13
Maximizing Data Source Stack Performance	4-14
Controlling a Procedure's Flow	4-15
Executing Other Maintain Procedures	4-15
Calling a Maintain Procedure on a Different Server	4-16
Passing Variables Between Procedures	4-17
Accessing Data Sources in the Child Procedure	4-19
Data Source Position in Child Procedures	4-19
The Advantages of Modularizing Source Code Using CALL	4-20
Optimizing Performance: Data Continuity and Memory Management	4-20
Executing External Procedures	4-21
Forms and Event-driven Processing	4-23
How to Use Forms	4-25
Designing a Form	4-25
Designing Event-driven Applications	4-26
Creating Event-driven Applications	4-26
Reading From a Data Source	4-26
Repositioning Your Location in a Data Source	4-27
Writing to a Data Source	4-28
Evaluating the Success of a Simple Data Source Command	4-28
Evaluating the Success of a Stack-based Write Command	4-29
Transaction Processing	4-30
Classes and Objects	4-31
What Are Classes and Objects?	4-31?

CHAPTER 1

Introducing WebFOCUS Maintain

Topics:

- Road Map: Where Should You Go?
- What Is WebFOCUS Maintain? (for New Users)
- Overview of Developing WebFOCUS Maintain Applications

The following topics offer introductory information about WebFOCUS Maintain for new users.

These topics also provide an overview of the step-by-step process for developing an application, from creating the front end to deploying the application.

Road Map: Where Should You Go?

Welcome to WebFOCUS Maintain!

- If you are new to WebFOCUS Maintain, we recommend that you read *What Is WebFOCUS Maintain? (for New Users)* on page 1-2 and then work through Chapter 2, *The WebFOCUS Maintain Basic Tutorial*.
- If you used WebFOCUS Maintain Version 4, we recommend that you read the *WebFOCUS Summary of New Features*.
- If you used Cactus Version 3, we recommend reading Appendix B in *Developing WebFOCUS Maintain Applications* to learn about the improvements we have added to WebFOCUS Maintain 5. Once you read this section, you can easily start developing applications. To see the most recent changes, see the WebFOCUS Maintain Read Me file.
- Finally, if you are interested in a quick overview of how to develop a WebFOCUS Maintain application, read *Overview of Developing WebFOCUS Maintain Applications* on page 1-4.

What Is WebFOCUS Maintain? (for New Users)

WebFOCUS Maintain is an application development tool that creates Web-based data maintenance applications to be deployed across an enterprise (which can be loosely defined as a company's computer systems). Using WebFOCUS Maintain and its multi-platform 4GL language, you can easily create, test, and deploy complex business applications that span the Internet, IBM mainframes, midrange servers, and workstations.

The Challenge of Accessing Information

If your company is like most companies, you probably have several different computing systems in operation. For example, your accounting and payroll systems are running on a midrange UNIX computer, while your inventory system is running on an IBM mainframe. Most of the employees have personal computers on their desks and are accustomed to easy-to-use point and click interfaces.

With WebFOCUS Maintain, you can develop applications with graphical front-ends that run on personal computers and access data on any computer system in your company.

How WebFOCUS Maintain Works

Using the Form Editor, you develop the application's front-end (or user interface). Then, using the Maintain Development Environment, you develop the code that extracts data from your data sources and updates the data sources with new data. WebFOCUS Maintain applications can also link to existing COBOL, C, or other 3GL programs, CICS transactions, IMS transactions, RDBMS stored procedures, and FOCUS procedures.

At run time, end users start the application and access a WebFOCUS Server where the data or procedures reside. The application extracts data from the data source, displays it for the end users to see, updates the data source with new information, and executes any procedures.

N-Tier Applications

We call WebFOCUS Maintain applications *n-tier applications* because they are capable of distributing processing over many platforms. N-tier applications offer the following advantages:

- **You can access data on multiple platforms**, thus forming relationships among disparate data sources.
- **Your application logic runs on the machine most capable of performing it.** For example, PCs are ideally suited for displaying the user interface of your application. On the other hand, MVS machines may not be capable of displaying pretty pictures, but they pack plenty of processing power. A WebFOCUS Maintain application's front-end runs on a PC, whereas the data access code can run on an MVS machine.
- **You can speed up your applications.** Procedures that access data can run on the platform where the data resides, thus ensuring that any aggregation or screening takes place immediately. This means that your application is not shipping large quantities of data across a network to be aggregated or screened somewhere else. Less network traffic means increased application speed.

Leveraging the Power of the WebFOCUS Server

The WebFOCUS Server is available for every major operating system—MVS, UNIX (several flavors), Windows, Open VMS, CICS, and VM/CMS and, using its data adapters, can access just about every major database management system, including: Informix, Sybase, DB2, and Ingres. At last count, we were up to over 60. The WebFOCUS Server can also execute procedures written in another language (such as COBOL, C, or other 3GL programs, CICS transactions, IMS transactions, RDBMS stored procedures, and FOCUS procedures), so you don't need to discard previous investments in software.

Because Information Builders has already worked out the complications of different operating systems, communications protocols, and data access languages, you, as a developer or end user, do not need to worry about or even know where your data is coming from.

In Summary: What WebFOCUS Maintain Can Do for You

Using WebFOCUS Maintain, you can:

- Easily develop data maintenance applications that can be run in both Web-based and Windows deployments with no prior knowledge of HTML, Java™, or complex 3GLs.
- Access and update data on every major operating system and every major database system (using the power of the WebFOCUS Server).
- Access and update data from different platforms at the same time (for example, the inventory system on MVS and the accounting system on UNIX).
- Preserve your investments in existing software since WebFOCUS Maintain can easily execute COBOL, C, or other 3GL programs, CICS transactions, IMS transactions, RDBMS stored procedures, FOCUS procedures, and many more.
- Take advantage of the strengths of your computing systems while sidestepping the weaknesses by partitioning your application among the platforms that can support them.
- Integrate seamlessly with Java, Java applets, JavaScript, and VBScript.

What's Next?

For more information on how to use WebFOCUS Maintain, we recommend working through Chapter 2, *The WebFOCUS Maintain Basic Tutorial*.

If you want to develop your own WebFOCUS Maintain applications immediately, we recommend reading *Overview of Developing WebFOCUS Maintain Applications* on page 1-4.

Overview of Developing WebFOCUS Maintain Applications

Now that you know what a WebFOCUS Maintain application is and how it works, you're ready for a step-by-step view of the development process. This section summarizes the steps:

1. Create a project and initial procedure.
2. Describe the data.
3. Logically partition the application.
4. Create the front end.
5. Create the data access logic.
6. Set up front-end and data access procedures to call each other.
7. Include other application components.
8. Test all the application's procedures locally.

9. Set up the deployment scenario.

10. Deploy and run the application to test it on the network.

A good way to get started immediately developing WebFOCUS Maintain applications is by using Update Assist. All you need is a Master File for a data source that you want to create an application for. For more information, *Developing Reporting Applications*.

The rest of this section is under development.

CHAPTER 2

The WebFOCUS Maintain Basic Tutorial

Topics:

- Before You Begin
- Creating a Project
- Familiarizing Yourself With the Explorer
- Adding a Data Source to Your Project
- Familiarizing Yourself With the Project Explorer
- Designing a Form
- Saving Your Work
- Running Your Project Locally
- Using Radio Buttons
- Giving Your Form a New Title
- Writing Data to the Data Source
- Adding a Form to Display Data From a Data Source
- Using the HTML Overflow Property
- Adding Form Navigation Buttons
- Adding Images to Your Project
- What's Next?

Welcome to the WebFOCUS Maintain basic tutorial! In this chapter, you are going to create a WebFOCUS Fan Club application. This application will enable you to maintain a data source of WebFOCUS Maintain “fans.” You will be able to add fans and view existing fans. This tutorial assumes no prior WebFOCUS Maintain or Maintain experience.

Before You Begin

Before you begin working on the WebFOCUS Fan Club application, make sure you have done the following:

- Installed and verified a WebFOCUS Server.
- Installed WebFOCUS Desktop Developer Studio on your machine (this includes the Maintain Development Environment).
- Verified your machine's connection to the WebFOCUS Server.
- Installed the sample tutorial files required for the WebFOCUS Maintain tutorial on your machine. This includes the following files:
 - The Fannames data source (both the .MAS and .FOC files).
 - The addafan.gif, currfan.gif, fan.gif, and spiralbg.gif image files.

Creating a Project

Before you can do any development work in WebFOCUS Maintain, you will need to create a project in WebFOCUS Developer Studio. The WebFOCUS Developer Studio *project* contains your working files. This is where you develop, test, and edit your code. When you place your project files on a WebFOCUS Server (called deploying) your project becomes an *application*.


You can create the project with the Create a Project Wizard. It will guide you swiftly through the project creation process by prompting you for all necessary information.

After you create the project, you will need to create a Maintain procedure in the project.

Procedure How to Create the Project Folder

1. Click the Windows *Start* button.
2. Select *Programs* from the Start menu.
3. Select *WebFOCUS Developer Studio* from the WebFOCUS 52 Developer Studio folder to start WebFOCUS Developer Studio.

WebFOCUS Developer Studio starts.

4. Open the Maintain Development Environment by clicking the *Open Maintain*  button.
5. In the Maintain Development Environment, click Projects on localhost in the Explorer window.

6. Do one of the following:

- Right-click Projects on localhost and click *New Project...*
- Click *New Project...* in the File menu.

The Create a Project - Step 1 of 2 dialog box opens.

Create a Project - Step 1 of 2

A project is a directory in which you store your procedures, file descriptions and other files that are necessary for a complete data analysis or reporting systems.

Please enter name for this project:

It is recommend that every project have its own directory. By default, the name of the project (specified above) is also name of the project's directory. You can enter a different name for this directory or browse for an existing directory.

Please specify?

Browse

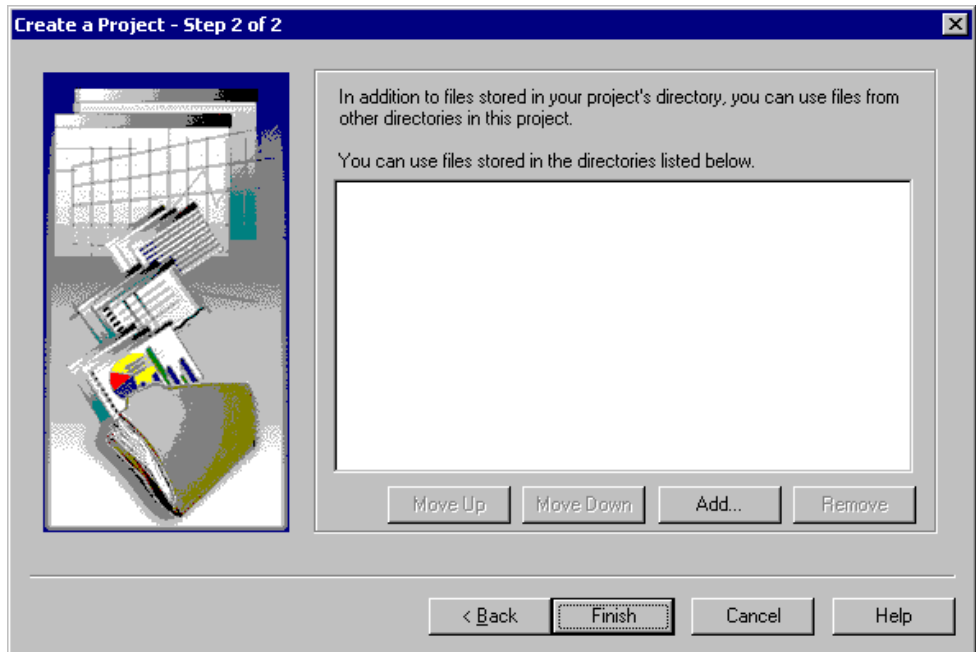
Location and name of your new project...

IBFS:/localhost/DEV

< Back Next > Cancel Help

- 7.** In the first input box, enter FanClub as the name of your new project. The wizard adds the extension .GFA and appends the file name to the default directory path that appears in the second input box.
- 8.** In the second input box, you can verify the directory path, or click *Browse* to select another directory name from the Choose Directory dialog box.
- 9.** Click *Next* to continue.
- 10.** A message is displayed if the directory does not exist. Click *Yes* to make the new folder. (Click *No* to change the file name or to exit the wizard.)

11. The Create a Project - Step 2 of 2 dialog box opens.

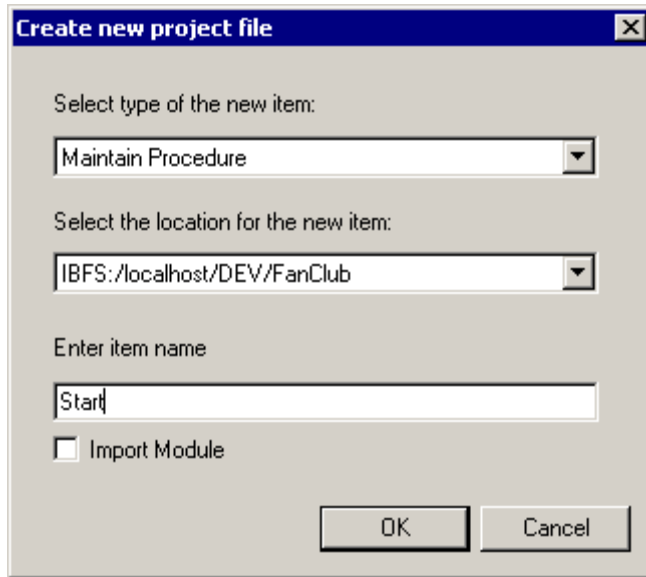


12. You don't need to do anything here, so click the *Finish* button. The new folder is added to the Explorer.

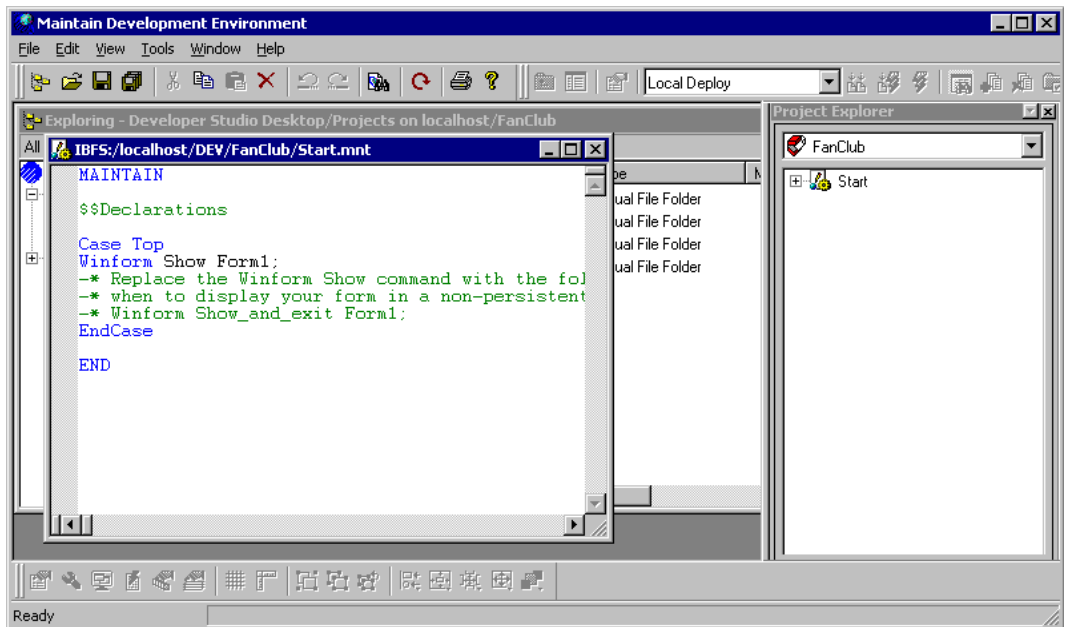
Procedure How to Create the Maintain Procedure

1. Right-click FanClub, click *New*, and then click *Project file*....
or
Right-click the Maintain Files folder in the FanClub folder, click *New*, and then click *Project file*....
2. In the Create new project file dialog box, enter the name of your procedure (let's use Start) in the *Enter item name* field.

3. Click OK.



You will see the following window:



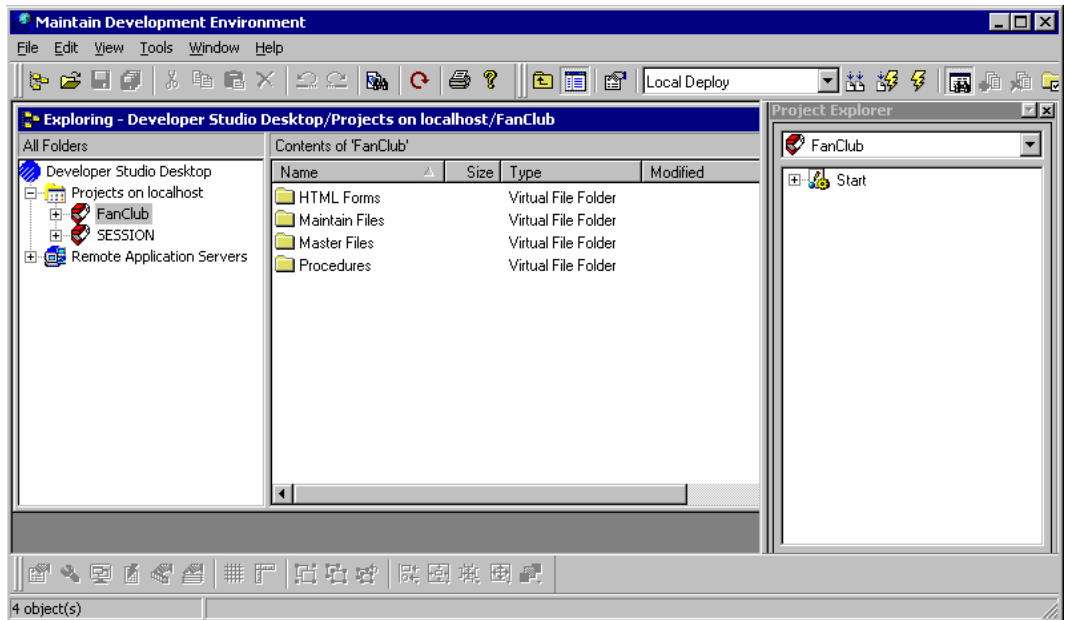
At the top of the window, underneath the menu bar, you will see toolbars with buttons for performing commonly used commands. (Later, you may wish to customize these toolbars, but for now we are going to leave them alone.)

Below the toolbars you should see the text editor, displaying the default code for Start; the Explorer window containing the contents of the FanClub project; and the Project Explorer, displaying another view of our project.

Familiarizing Yourself With the Explorer

The Maintain Development Environment displays two views of FanClub: in the Explorer window and in the Project Explorer window. For now, let's close the text editor containing the code for Start.mnt.mnt and concentrate on the Explorer.

(If you already know how to use WebFOCUS Developer Studio, the Explorer will already be familiar to you.)



The left pane of the Explorer displays the contents of the Developer Studio Desktop. The first item, Projects on localhost, displays the projects you are developing. FanClub, the project you created, is listed first, followed by SESSION, the default WebFOCUS project.

Under Projects is a list of Remote Application Servers, which list the WebFOCUS Servers available to you. Depending on how you have configured your computer, you will see one or more WebFOCUS Servers. We will learn more about these servers later when we deploy our project.

The right pane of the Explorer displays the contents of whatever you select in the left pane. It should be displaying the contents of the FanClub project. The default folders for a project are HTML Forms, Maintain Files, Master Files, and Procedures, but you can add new ones if you wish.

Frequently, you will be using previously existing files in a project, and the Explorer enables you to easily view all files available in the project path. You can also easily turn off this view to view only the files that have been included in the project.

Procedure How to View the Files Available to a Project

1. Double-click any of the folders in either Explorer pane except for Maintain Files (either HTML Forms, Master Files, or Procedures).
2. In the toolbar, click on *Display all files in the project path* button.



(If this button is not available, click *Show All Files* in the View menu.)

The default is for *Display all files* to be on, but your setting may be different.

As you turn *Display all files* on and off, you should see files appearing and disappearing in the right pane of the Explorer. These files are available for you to use in constructing your project, but none of them are actually included in the project yet.

Adding a Data Source to Your Project

The first step you take when designing a project is probably to define the data sources on which your final application is going to operate.

WebFOCUS Maintain applications can operate on data from many types of data sources, using the capabilities of the WebFOCUS Server. To do so, an application needs the following:

- An Information Builders data source description, which describes the structure of the data: what the fields are, what their format is, and so on, and where to find the data.
- Data sources that contain the actual data. WebFOCUS Maintain applications cannot use these data sources without the *data source* description telling them how the data is organized.

To define which data sources your application is going to operate on, you include its data source description in your project. There are two ways to get a data source description into your project:

- Create one from scratch.
- Use an existing one.

Tip: The WebFOCUS Server ships with utilities that create data source descriptions (even from other database management systems, such as Oracle). In general, the only reason you need to create data source descriptions from scratch is if you are creating a brand new data source.

For this tutorial, we will be using existing data source descriptions.

Procedure How to Add a Data Source to Your Project

1. If you have not already done so, open the Master Files folder in the Explorer and turn on *Display all files*.
2. Find the fannames.mas data source (you may need to adjust the size of the window so that you can view the scroll bars).
3. Right-click fannames.mas and click *Add to Project*.

Note that the icon to the left of fannames.mas is now displayed in color, while the rest are in black and white.

If you turn off *Display all files*, you will see only fannames.mas.

Familiarizing Yourself With the Project Explorer

The Project Explorer, which appears on the right side of your screen, displays another view of your project.

Currently, the only component here is Start, which you created in WebFOCUS Desktop Studio, but as you continue working on the Fan Club project, you will see more components here.

If you expand Start, you will see the following:




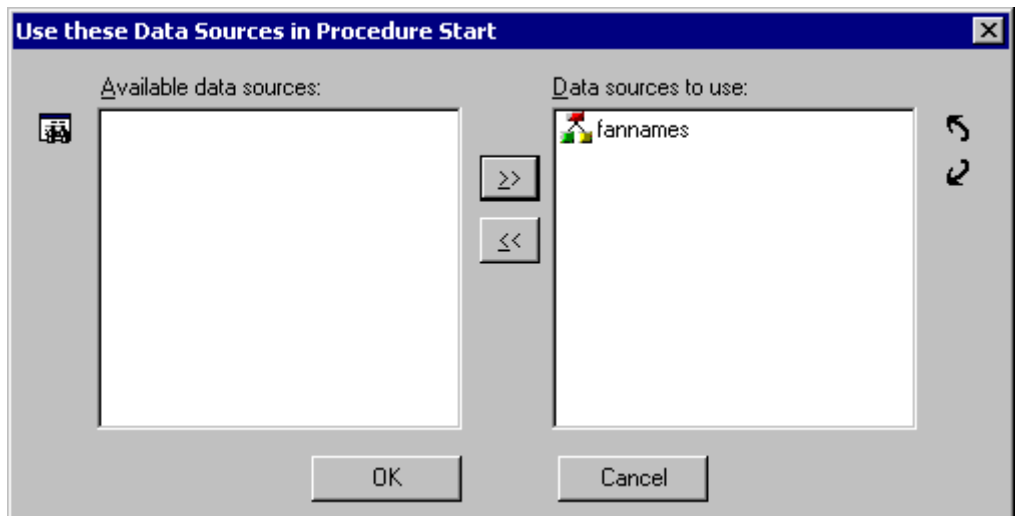
There is a folder for all of the forms in Start; currently there is one form here, Form1. If you expand it (not open it), you will see the text "There are no items to show." This form was created by default when you created Start.

The next component is called Top. Top is a *function*, which is a series of commands in a procedure grouped together as a unit of control. A function accomplishes some small task, such as calculating values, extracting data from a data source to place in a data source stack, or writing information to a data source.

Every procedure has a Top function, which consists of the commands that will be executed immediately when the procedure is executed. WebFOCUS Maintain automatically creates the Top function when you create a Maintain procedure.

Procedure How to Use a Data Source in a Procedure

1. In the Project Explorer, right-click Start, and then click *Use data sources....*
2. In the Use these Data Sources in Procedure Start dialog box, select fannames in the list of available data sources.
3. Click  to move fannames into the list of Data sources to use.



4. Click OK.

More About the Project Explorer: Viewing the Structure of a Data Source

Once you use the Fannames data source in the Start procedure, its name appears in the Project Explorer underneath the Data Sources folder with a plus sign next to it.

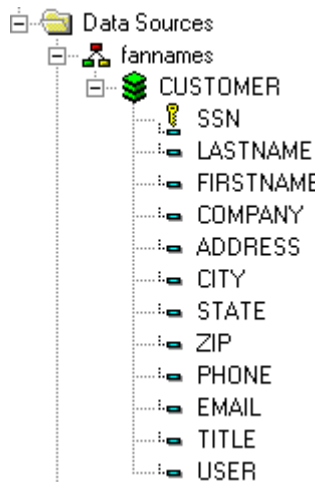
If you click the *plus* sign next to Fannames, WebFOCUS Maintain displays the segments in the Fannames data source:



A *segment* is a collection of fields that have a one-to-one relationship to each other. The Fannames data source has one segment, called CUSTOMER.

Note: This data source has only one segment. Hierarchical data sources, such as FOCUS, can have more than one segment. If you would like to see a data sources with more than one segment, migrate the CAR data source description (one of Information Builders' standard sample files) into your project and view it. You can then delete the CAR data source description from your project by selecting it and pressing the Delete key.

If you click the plus sign next to CUSTOMER, you will see the fields in that segment.

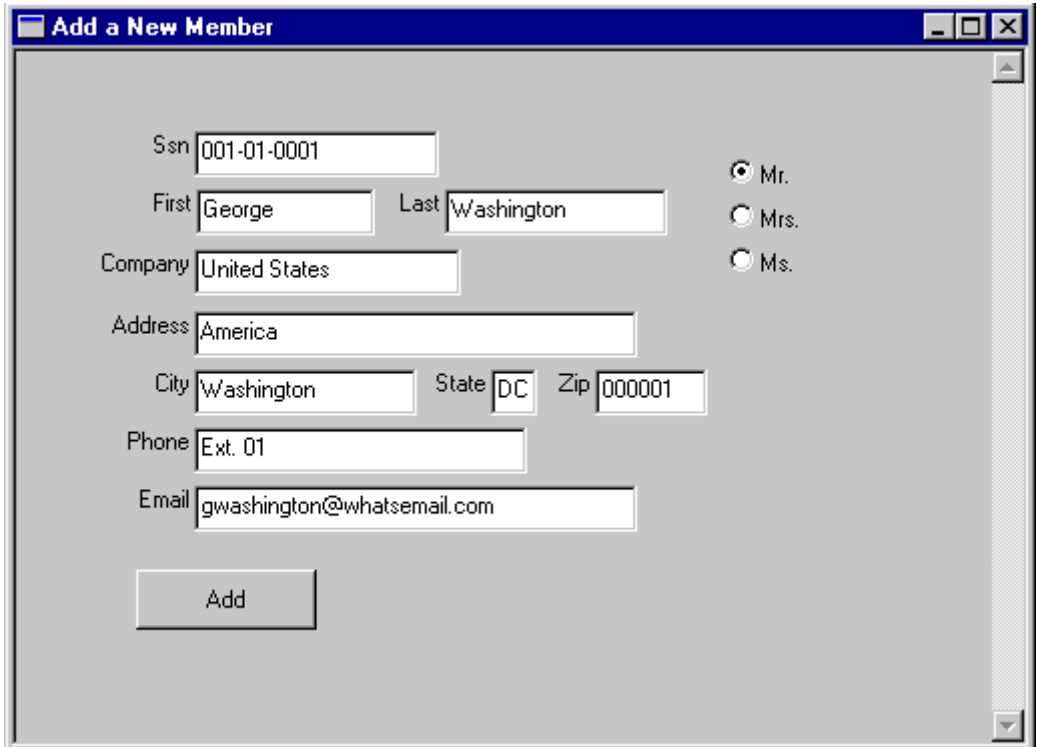


Since Fannames has only one segment, these are the only fields in the Fannames data source.

The key next to the SSN field indicates that SSN is a key field, that is, it uniquely identifies the segment instance. In the Fannames data source, each fan has one SSN, and no other fan should have the same number.

Designing a Form

The first form that you will create for the Fan Club application will look something like this:

A screenshot of a web form titled "Add a New Member". The form contains several input fields: "Ssn" with the value "001-01-0001", "First" with "George", "Last" with "Washington", "Company" with "United States", "Address" with "America", "City" with "Washington", "State" with "DC", "Zip" with "000001", "Phone Ext." with "01", and "Email" with "gWASHINGTON@whatsemail.com". To the right of these fields are three radio buttons labeled "Mr.", "Mrs.", and "Ms.", with "Mr." selected. At the bottom left of the form is an "Add" button. The form has a blue title bar and standard window controls.

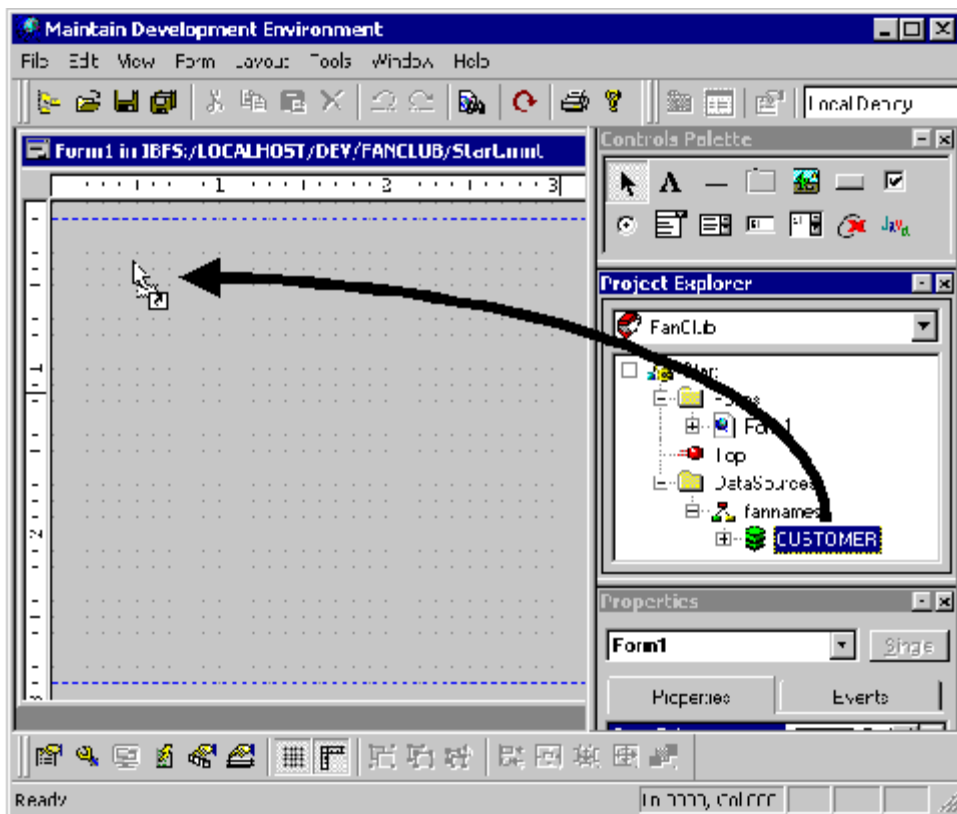
This form enables end users to add a new fan to the data source (yourself, perhaps?).

The information entered in this form will be written to the Fannames data source, so it should correspond to the fields in the data source. The fastest way to create the fields on the form is to use the existing data source fields. Therefore, your first task is to add data source fields to the form.

Procedure How to Add Data Source Fields to a Form

1. Open Form1.
2. In the Project Explorer window, click the *plus* sign to the left of Fannames to view the CUSTOMER segment.

3. Click and drag the CUSTOMER segment into the form.



WebFOCUS Maintain opens the Select Segment Fields dialog box. You will need to specify which data source stack fields you want to display on your form. First, let's discuss the role of data source stacks in your project.

What Are Data Source Stacks?

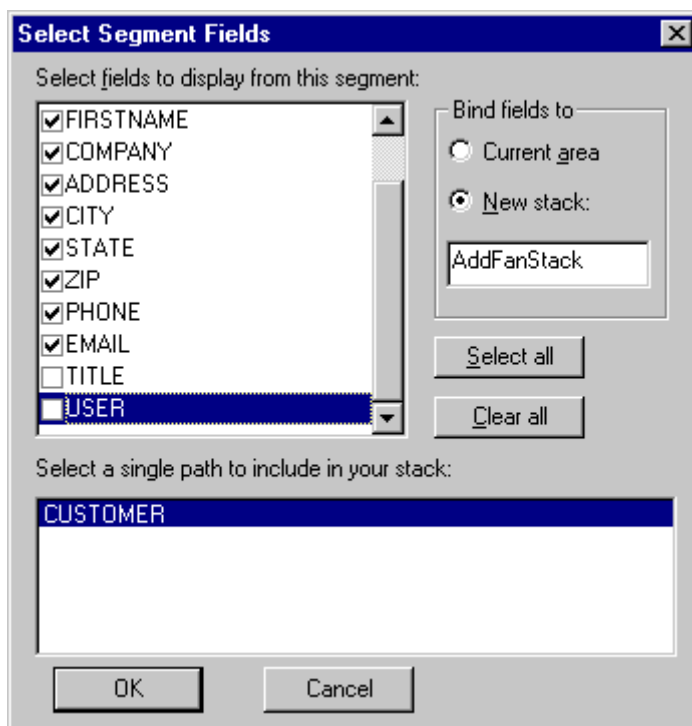
WebFOCUS Maintain procedures do not display or manipulate information in a data source directly. Instead, they use data source stacks as intermediaries between end users and the data source.

A *stack* is a non-persistent (or in-memory) table where you can store and manipulate data from one or more data sources. WebFOCUS Maintain procedures use stacks to hold values you read from the data source and to manipulate data before writing it back to the data source.

Since this is a new procedure, there are no stacks yet, so we're going to create one here named *AddFanStack*, instead of using the default *CustomerStk*. The structure of the stack is based on the fields in the data source—in other words, the stack is going to have the columns *SSN*, *LASTNAME*, *FIRSTNAME*, and so on. However, this stack is not going to have any data in it until we do something to put data in it.

Procedure How to Specify a Data Source Stack for Fields on a Form

1. In the Select Segment Fields dialog box, select the *New Stack* option. This specifies where your procedure will store the data that will be displayed in the fields. An input field appears under the *New Stack* radio button.
2. Type *AddFanStack* in the New Stack field.
3. Scroll down to the bottom of the list of fields and deselect *TITLE* and *USER*.



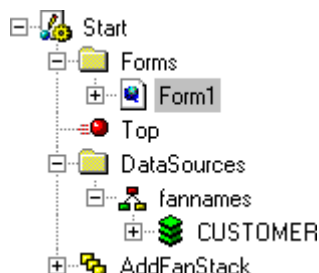
4. Click *OK*.

WebFOCUS Maintain places the first ten fields of the *CUSTOMER* segment into the form.

More About the Project Explorer: More About the Components of a Procedure

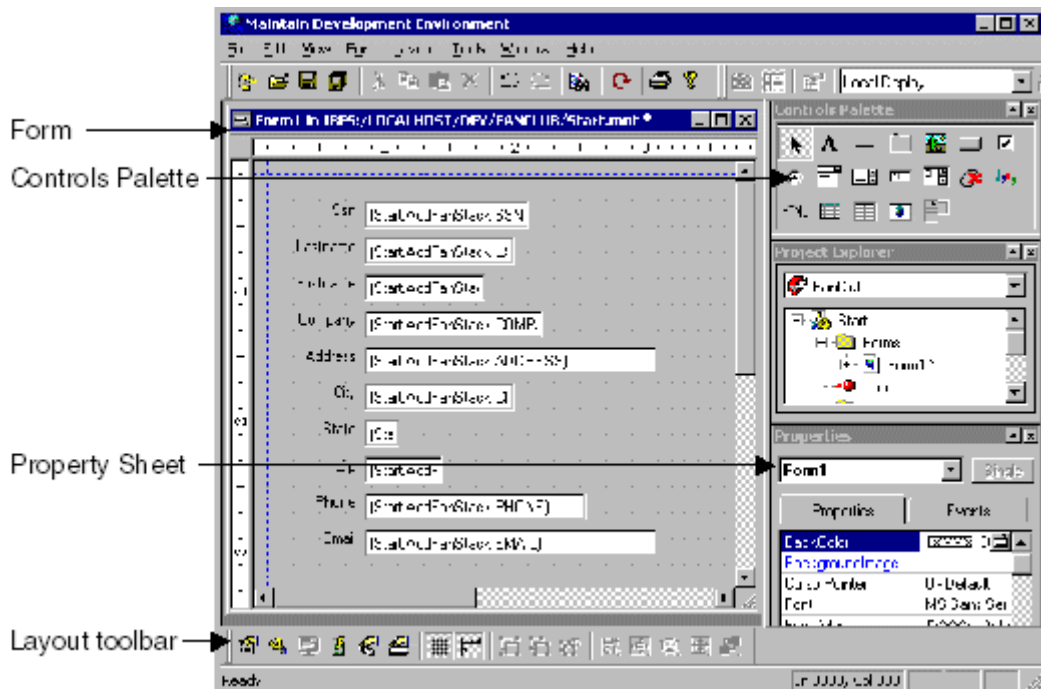
If you look in the Project Explorer, there have been some additions to the components of Start:

- If you expand Form1, you will see a list of fields that you placed on the form.
- The stack AddFanStack has also been added to Start. If you expand AddFanStack, you will see the columns in this stack.



Working in the Form Editor

The Form Editor is where you design the user data adapters for your applications. Let's take a look at its contents.



Note: The screen may look different from what you see here. The windows have been moved around.

- The Form is your main working area. What you place here is what appears on the form your end users see.
- The Maintain Controls palette contains the controls that you can place on your form, such as text boxes, buttons, radio buttons, check boxes, and so on. (Controls are objects on the screen that you can maneuver to execute an action.) You will have the opportunity to use many of these controls in this tutorial.
- The property sheet is where you specify information about controls that you have placed on the palette. You always see the properties for the currently selected control here. If no control is selected, you see the properties for the form itself.
- The Layout toolbar contains alignment and grouping commands.

Moving Controls on a Form

When we placed these fields on the form, WebFOCUS Maintain placed them in a column. This is not an optimal arrangement for these fields. Let's rearrange them so that Firstname and Lastname are next to each other in a row, and City, State, and Zip are also next to each other.

Procedure How to Move Controls on a Form

1. If the grid for your form is not on (the default setting), then turn it on either by selecting *Grid* from the View menu or by clicking the *Toggle grid* button on the Layout toolbar. Make sure the form is the active window.

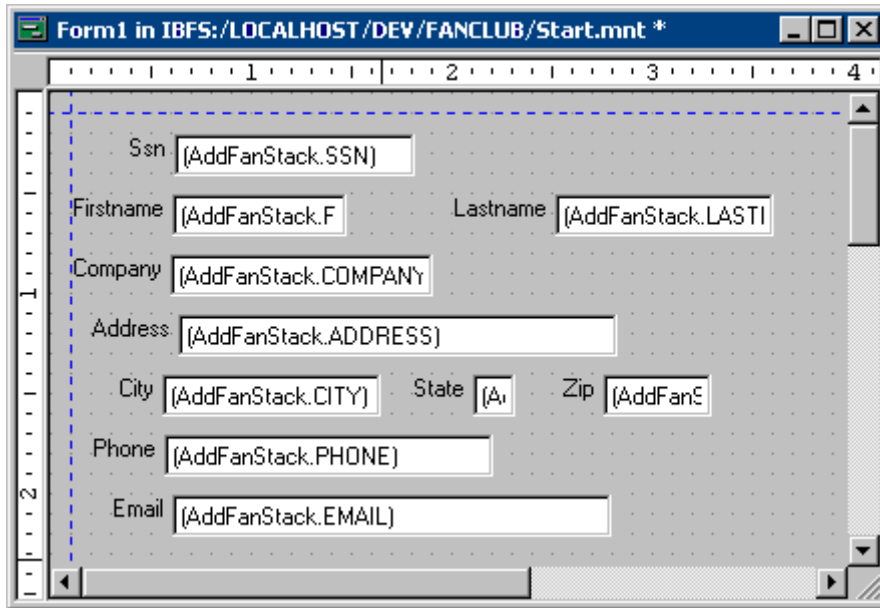


If the grid is on, when you move controls around on your form the Form Editor automatically snaps them to the closest grid point. This feature enables you to keep controls on a form aligned more easily.

2. Move Lastname (by clicking on it and dragging it to the new location) far enough to the right so that you can move Firstname up and to the left of it.
3. Move up the fields Firstname, Company, Address, and City.
Since the grid is on, you will preserve the correct vertical spacing between the fields.
4. Click and drag State to the right of City.
5. Click and drag Zip to the right of State.
6. Move up Phone and Email.

7. Right-click the form. From the pop-up menu, select *Tab Order* and reorder the fields to reflect the order in which they appear on your form.

When you are done, your form will look something like this:



Renaming Field Prompts

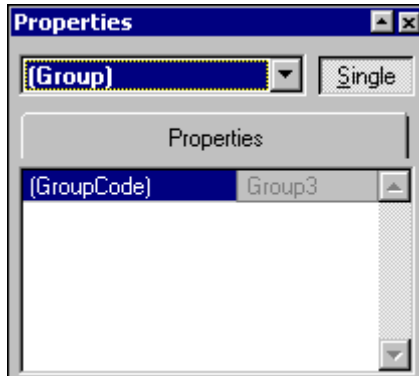
While Firstname and Lastname are certainly descriptive field prompts, they are definitely not English words. Let's rename these two field prompts to "First" and "Last."

Procedure How to Rename Field Prompts

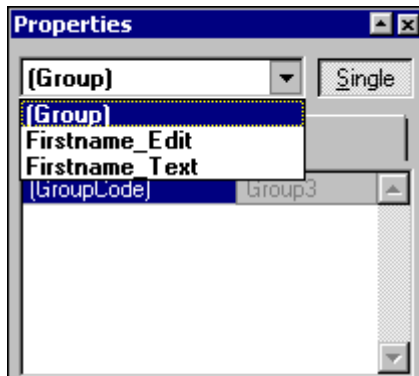
1. Select the Firstname field.

Notice that the control in the property sheet says (Group), indicating that the Firstname field is not a single control but is actually two controls grouped together. (It is useful to group controls so that WebFOCUS Maintain will treat them as a unit. This means that you can't move one without moving the other.)

Firstname is made up of two controls: a text box named Firstname_Text that serves as the prompt and an edit box control named Firstname_Edit where the end user will enter a value for the Firstname field. (Together this is called a prompted edit box.)



2. Select *Firstname_Text* from the drop-down list at the top of the property sheet.



You can then see the properties for Firstname_Text.

3. Scroll to find the Text property (the value will be *Firstname*), and change the value to *First*.



4. Press the Enter key to confirm your change.

Procedure How to Rename Field Prompts the Fast Way

There is a faster way to rename field prompts. Let's rename Lastname the fast way.

1. Select the Lastname prompted edit box.
2. Place your cursor over the Lastname prompt.
It will turn into a text editing I-beam.
3. Change the prompt to *Last*.

Aligning Controls

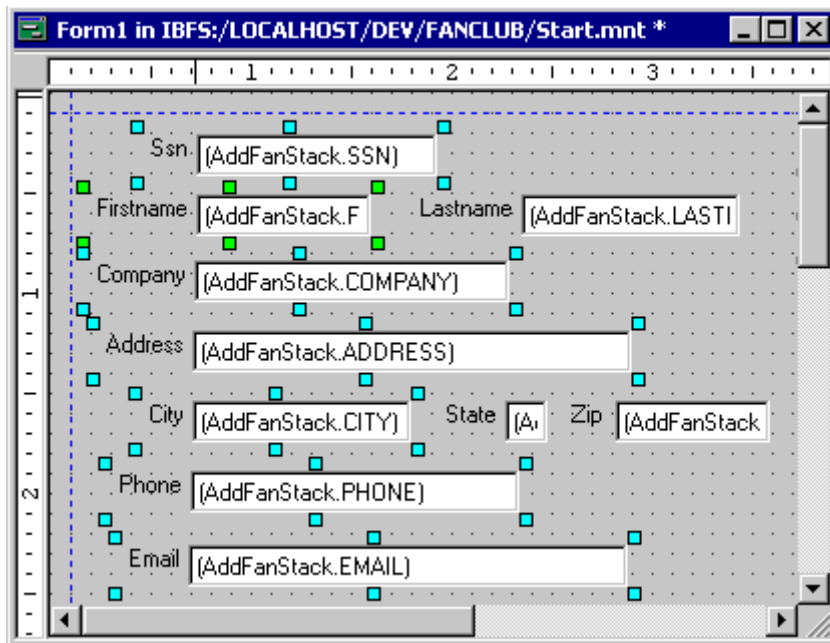
If you have been moving these controls around in the window and renaming prompts, your fields have probably gotten misaligned. The Form Editor enables you to align controls easily.

One of the alignment tools is designed especially for aligning entry fields such as these, which are made up of a text control and an edit box control.

Procedure How to Align Entry Fields

1. Select all of the entry fields on the left side (that is all of the entry fields except for Last, State, and Zip).
You can multi-select fields by selecting the first one and then holding down the Shift key or the Control key while you select the remaining fields.

Notice that one of the fields has green handles, while the other ones have blue handles. The field with green handles is the anchor control —that is, the field that all the other fields will be aligned with.



2. In the Layout menu, click *Align*, and then click *Center vertically* in the submenu.

or

Click the *Align Edges* button in the Layout toolbar.



Then click the *Center vertically* button in the pop-up toolbar.



3. Your entry fields should look like this:

Form1 in IBFS:/LOCALHOST/DEV/FANCLUB/Start.mnt *

Ssn. (AddFanStack.SSN)

Firstname (AddFanStack.F) Lastname (AddFanStack.LASTI)

Company (AddFanStack.COMPANY)

Address (AddFanStack.ADDRESS)

City (AddFanStack.CITY) State (A) Zip (AddFanStack)

Phone (AddFanStack.PHONE)

Email (AddFanStack.EMAIL)

Saving Your Work

Before you go any further, you should save your work.

Whenever you make changes to a component, WebFOCUS Maintain places an asterisk next to the component's name in the Project Explorer and in the title bar.

***Procedure* How to Save Your Project**

1. Do one of the following:
 - In the File menu, click *Save*.
 - Press Ctrl+S.
 - Click the *Save* button in the General toolbar.



Running Your Project Locally

Let's try running your project to see what it looks like.

Note: Whenever you deploy or run your project, WebFOCUS Maintain saves it.

Procedure How to Run Your Project Locally

To run your project, do one of the following:

- In the File menu, click *Deploy*, then click *Deploy and Run*.
- Click the *Deploy and run* button on the General toolbar.



WebFOCUS Maintain checks your project for errors and then opens the final application.

A screenshot of a web browser window titled 'Untitled'. The window displays a form with several input fields. The fields are labeled: 'Ssn', 'First', 'Last', 'Company', 'Address', 'City', 'State', 'Zip', 'Phone', and 'Email'. Each label is followed by a text input box. The 'State' field is a small dropdown menu. The form is set against a light gray background.

Our application enables us to enter information into the fields displayed in the window (which places data into the data source stack `AddFanStack`), but this is all we can do. Let's improve the functionality and appearance of our application.

For now, click the close box located on the upper right hand corner of the application to end it before continuing.

Notice that WebFOCUS Maintain has added a new folder, `DEPLOY`, to the list of folders under the project in the Explorer.

Using Radio Buttons

When we dragged the fields from the data source into the form, we deselected the TITLE and USER fields, which meant that these two fields are not in the form.

The reason we did not copy TITLE into the form is that we do not want end users to be able to enter whatever they want in this field; instead we want them to enter either Mr., Mrs., or Ms. We will do this using a radio button control. This prevents them from entering titles like Lord, Queen, or Your Most Honorable Excellency.

We have split the task of using radio buttons into three steps:

1. Adding the group of radio buttons to your form.
2. Adding tool tip text to the group of radio buttons. This step is optional, but it demonstrates how you can display useful information to your end users.
3. Binding the results of the selection to a stack.

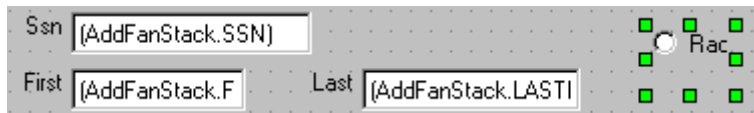
Procedure How to Add a Group of Radio Buttons to Your Form

1. Click the *Radio button* control in the Maintain Controls palette.

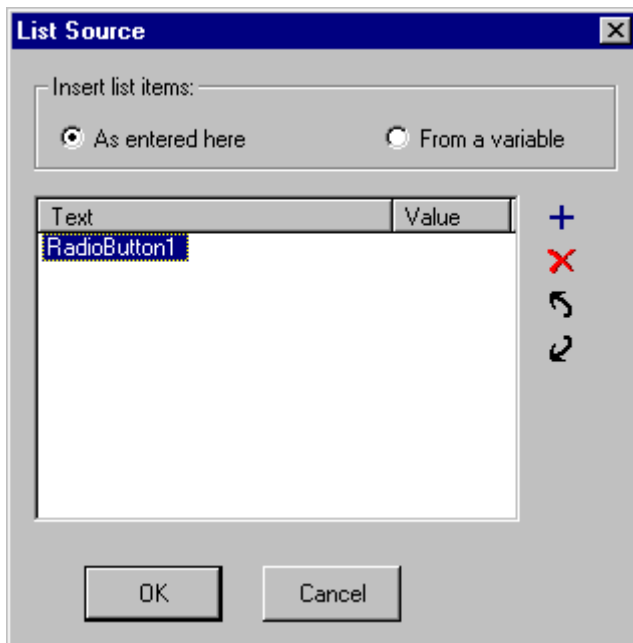


2. Draw a small rectangle on the form next to Ssn, First, and Last.

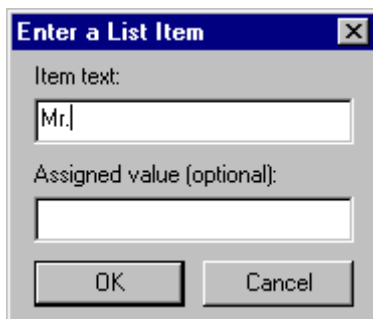
When you let go of the mouse button, you will see the green handles indicating that the control is selected. If you drew the rectangle large enough, you might see one radio button and the description of this radio button. (If you see nothing, do not worry; you can enlarge the size of the control.)



3. Double-click the control to open the List Source dialog box:



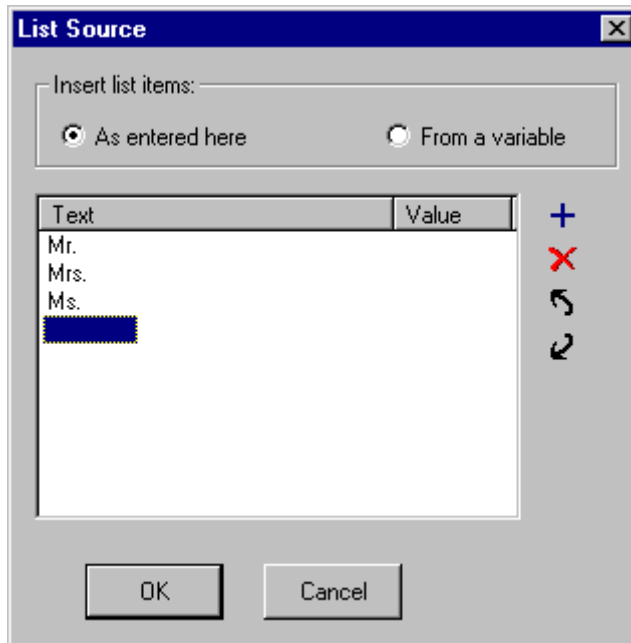
4. Double-click the highlighted text *RadioButton1* to open the Enter a List Item dialog box.
5. Replace the text *RadioButton1* with the text *Mr.* You do not need to enter anything in the Assigned value field.



6. Click *OK*.
7. Insert a new value by pressing the down arrow key.
8. Press the F2 key to edit the new value directly.
9. Type *Mrs.* and press the Enter key.

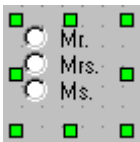
Pressing Enter confirms your new value and creates a new value entry area.

10. Repeat steps 8 and 9 to enter the value *Ms.*



11. Click OK.

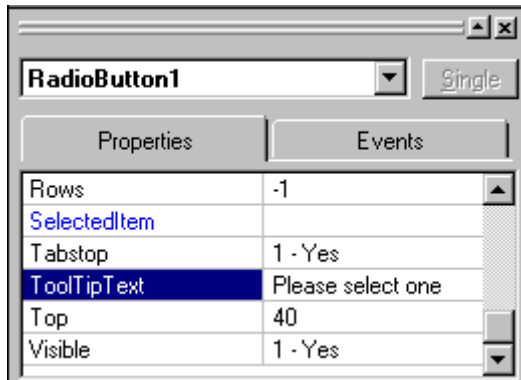
Your radio buttons should look like the following (you may need to adjust the size of your control to see everything):



Procedure How to Add Tool Tip Text

1. Make sure you have selected your group of radio buttons.
2. In the Properties sheet, click the *Properties* tab.
3. Select the *ToolTipText* property.

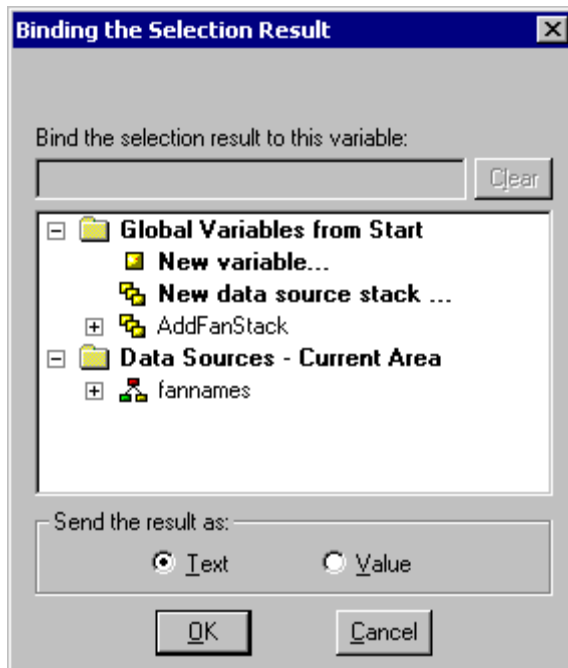
4. In the empty field to the right, type *Please select one* and press Enter.



You can move your cursor over the radio buttons in the Form Editor to see the tool tip.

Procedure How to Bind the Results of the Selection to a Stack

1. Make sure you have selected your group of radio buttons.
2. In the Properties sheet double-click the *SelectedItem* property to open the Binding the Selection Result dialog box. (You will probably need to scroll down to see this property.)



3. Expand *AddFanStack* by clicking the *plus* sign to its left.
4. Select the *TITLE* field.
5. Make sure *Text* is selected for *Send the result as*.
6. Click *OK*.

Now, when end users select either Mr., Mrs., or Ms. in this form, their choices are saved in the *TITLE* column of *AddFanStack*.

If you wish, save your work and deploy and run your application again to see what it looks like. Make sure to close the application before continuing the tutorial.

Stacks and Implied Columns

When you created *AddFanStack* using the Select Segment Fields dialog box (see *How to Add Data Source Fields to a Form* on page 2-11), you deselected the *TITLE* and *USER* fields. Since you deselected these fields, why are they showing up as columns in the data source stack?

These fields are showing up as columns in the data source stack because of the way WebFOCUS Maintain defines data source stacks. When you use a field in the data source to define a column in a data source stack, WebFOCUS Maintain defines columns based on the rest of the fields in that data source or, if your data source is hierarchical, the rest of the fields in the segment and the key fields in any parent segment.

WebFOCUS Maintain includes all of these fields in your data source stack so that when you update your data source from the data source stack, it knows the path to these fields.

These columns are called *implied columns*.

You may also have noticed that there is one field from the *Fannames* data source that you have not placed on the form: the *USER* field. The reason you have not placed it on the form yet is because the end user will not be entering this field; instead it will be generated by the application.

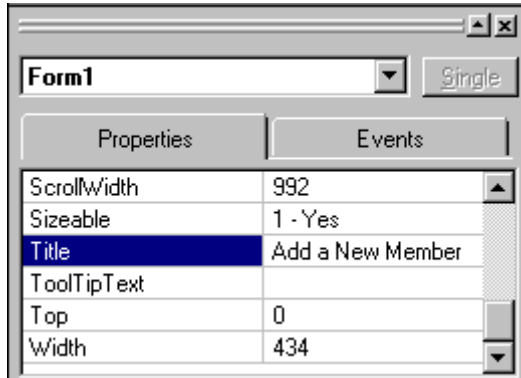
Giving Your Form a New Title

When you ran your application, you probably noticed that the title bar said "Untitled." Let's change this so that your application displays a more useful title at run time.

Procedure How to Give Your Form a Title

1. Deselect all controls in your form (that is, click anywhere in the form that is not a control).
2. In the property sheet, scroll down to the *Title* property and select it.

3. Change Untitled to *Add a New Member*.



If you wish, save your work and deploy and run your application to see what it looks like. Make sure to close the application afterwards before continuing the tutorial.

Writing Data to the Data Source

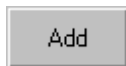
Your next step is to enable end users to write the data that they enter in this form to the data source. They will do this by clicking a button that says *Add* at the bottom of the form. You will add the button to your form and write the Maintain Language source code that inserts the data into the data source.

Procedure How to Add a Button to Your Form

1. In the Form Editor, select the *Button* control on the Control palette.



2. Move the cursor to the bottom of your form and draw a rectangle where you want your button to go.
3. Change the text in the button to *Add*. (The text in the button is automatically selected when you create the button, so all you need to do is type your new text.)



4. In the property sheet, notice that the first property is (Name) and its value is Button1. This is the internal program name of the button and is how WebFOCUS Maintain refers to it. By default, WebFOCUS Maintain names forms and controls with their type and a unique number. If you are planning to refer to a form or control in other places of the procedure, we recommend that you rename it to something more descriptive.
5. Change the (Name) of the button to *AddButton*.

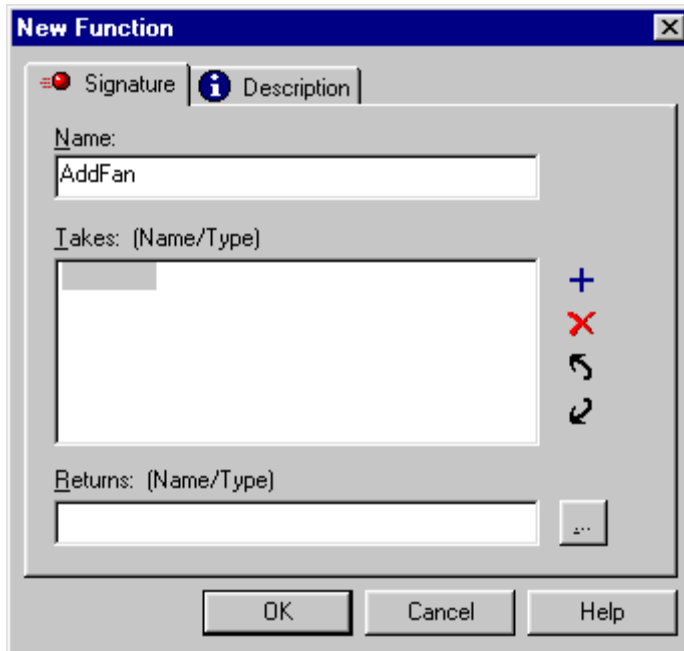
Writing Functions

Now that you have created a button that the end user will click, you need to create the code that will be executed when the end user clicks the button.

You will put this code in a function. A function is a series of commands in a procedure grouped together as a unit of control. A function accomplishes some small task, such as calculating values, extracting data from a data source to place in a data source stack, or writing information to a data source.

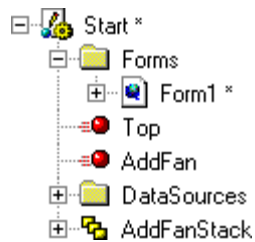
Procedure How to Write a Function

1. In the Project Explorer, select the *Start* procedure.
2. Right-click *Start* and, in the shortcut menu, click *New*. Then click *Function (Case...)* in the submenu.
3. In the New Function dialog box, give your function the name *AddFan*.



4. Click *OK*.

Your new function appears in the list of components of the Start procedure.



5. Double-click *AddFan*.

WebFOCUS Maintain opens the source code for your function in the Procedure Editor.

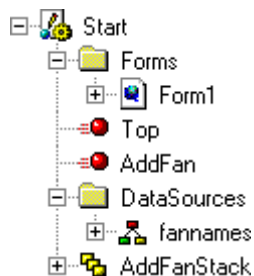
About the Procedure Editor

Underlying many of the graphical elements of your WebFOCUS Maintain project is Maintain language source code.

You can edit this source code directly using the Procedure Editor.

Let's take a brief look at the source code for the Start procedure, compared with the components that we have added so far in our application development process.

Note: The text in your window may wrap differently.



```

MAINTAIN FILE Fannames

$$Declarations

Case Top
Infer fannames.CUSTOMER.SSN into AddFanStack;
Winform Show Form1;
-* Replace the Winform Show command with the
following code
-* to display your form in a non-persistent
state
-* Winform Show_and_exit Form1;
EndCase

Case AddFan

EndCase

END
  
```

The Start procedure starts with the line

```
MAINTAIN FILE Fannames
```

and ends with the keyword

```
END
```

All WebFOCUS Maintain procedures start with the keyword MAINTAIN (which must be typed in uppercase) and end with the keyword END (also uppercase). When you create a procedure, WebFOCUS Maintain puts these two keywords in for you automatically.

The second and third words, FILE Fannames, tell WebFOCUS Maintain what data sources this procedure is going to access. If you look in the list of project components, you will see a Data Sources folder and, under this, the Fannames data source.

Following this line is a comment line beginning with \$\$\$. If you have syntax coloring on, you can tell this is a comment (green is the default color for comments, although you may have a different setting). You turn syntax coloring on and off using the Editor tab in the Options - Maintain Development Environment dialog box (in the Tools menu, click *Environment options...*). This comment line is automatically generated when you create a procedure.

This particular comment, \$\$\$Declarations, is generated automatically by WebFOCUS Maintain when you created the procedure. If you create any variables, WebFOCUS Maintain places their source code after this comment.

The next line, Case Top, begins the definition of the Top function. This definition takes up several lines and ends with the keyword EndCase. The first statement, which begins with Infer, defines the AddFanStack data source stack.

The next line of the Top function

```
Winform Show Form1;
```

is the code that displays Form1 at run time. This code was generated automatically when you created the procedure.

The next three lines begin with the characters -* and are also comments. They contain an alternate piece of code for opening a form:

```
Winform Show_and_exit Form1;
```

This code opens Form1 and then exits the application while still displaying Form1.

The final two lines before the END keyword that ends the procedure

```
Case AddFan
```

```
EndCase
```

are what define the function you just created. You are going to add some code to this function using the Language Wizard.

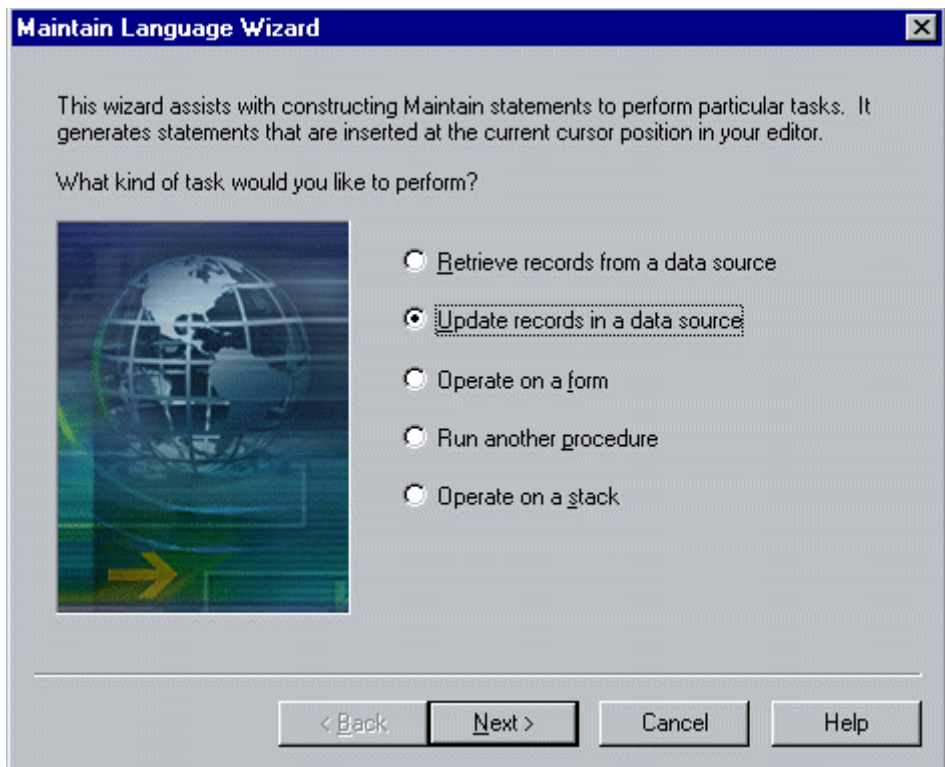
Procedure How to Build Maintain Language Code Using the Language Wizard

1. Ensure that your cursor is in the line between Case AddFan and EndCase.
2. Right-click in the *Procedure Editor* window.
3. From the shortcut menu, select *Language Wizard*.

The Maintain Language Wizard opens. The Maintain Language Wizard helps you build Maintain language source code without your having to type the syntax yourself.

The first Language Wizard window asks you to specify, in general, what kind of task you want to accomplish.

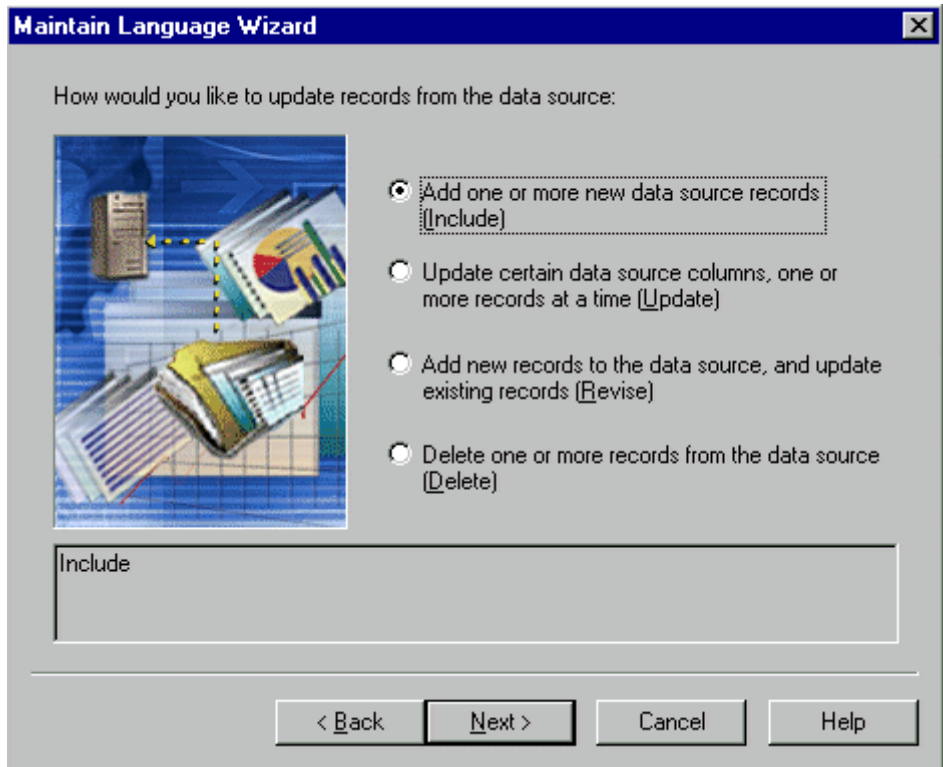
4. Select *Update records in a data source*.



5. Click *Next*.

Now that you have specified the general task you want to perform, the Language Wizard narrows down the task further. Notice that after each task, there is a word in parentheses. This is the name of the Maintain Language command that executes that task. Notice also the box at the bottom of the window contains the Maintain language code being generated by the Language Wizard. As you move through the Language Wizard, you will see more code here.

6. Select *Add one or more new data source records (Include)*.




7. Click *Next*.

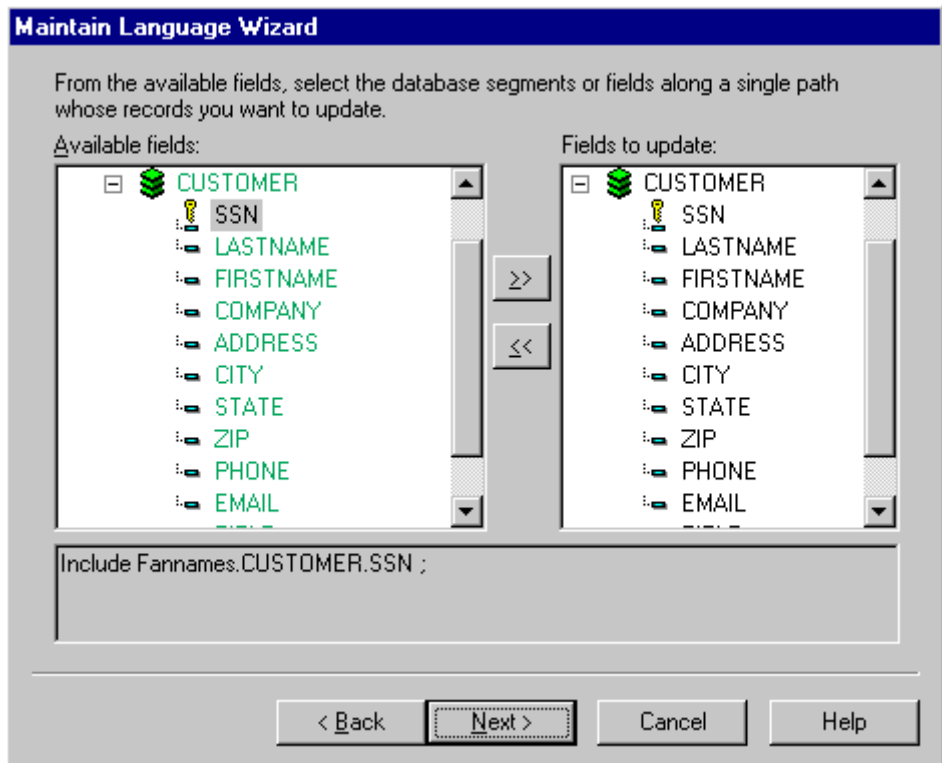
Now that you have specified which command to use, the Language Wizard asks you to supply the parameters for that command. In this case, you need to tell it which data source is being updated and from where.

You first specify which data source is being updated.

Note: The *Available fields* list contains the data sources that you are using in this procedure, not the list of data sources in the project!

8. Expand the *Fannames* data source.

9. Expand the *CUSTOMER* segment.
10. Copy the *SSN* field into the *Fields to update* box by clicking *SSN* and then clicking the  button, or by double-clicking *SSN*. Notice that, as with stacks, all the other fields in the *CUSTOMER* segment come along for the ride. (See *Stacks and Implied Columns* on page 2-26.) This is because the Maintain language assumes that if you are adding new data source records to a data source, you will want to write information into all the fields in a segment.



Notice that the Maintain language box at the bottom now reads:

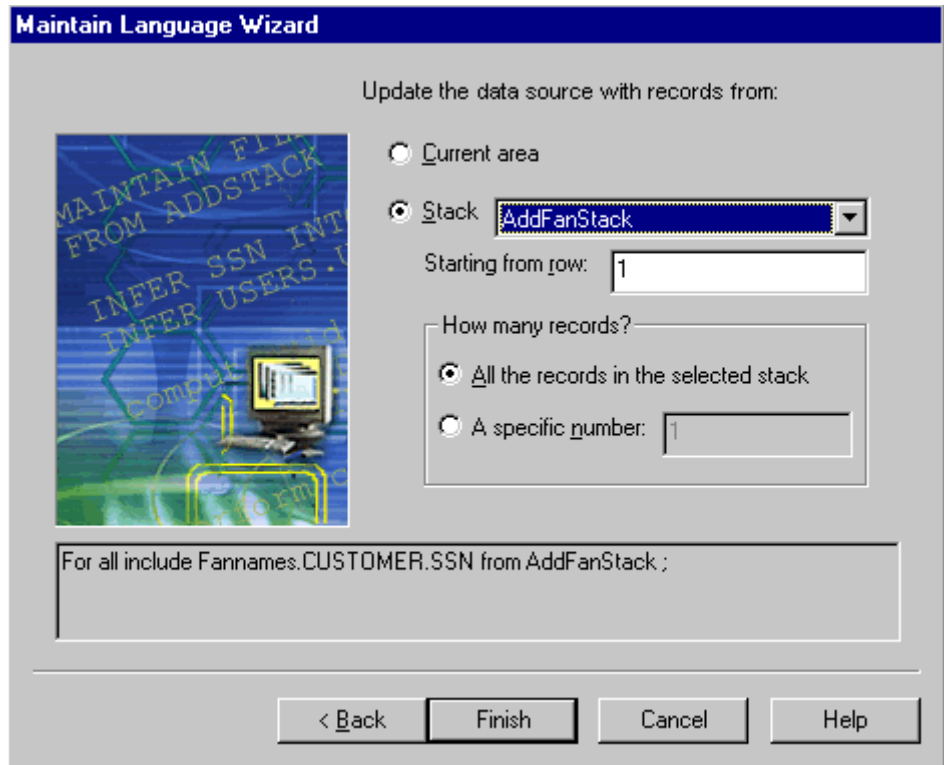
`Include Fannames.CUSTOMER.SSN;`

11. Click *Next*.

Your final step is to indicate where this data is being written from, which in this case will be *AddFanStack*.

12. Select *Stack*.
13. Choose *AddFanStack* from the list.

14. Leave the 1 in the Starting from row field and select the *All the records in the selected stack option*.



Maintain Language Wizard

Update the data source with records from:

☐ Current area

☒ Stack **AddFanStack**

Starting from row: **1**

How many records?

☒ All the records in the selected stack

☐ A specific number: **1**

For all include Fannames.CUSTOMER.SSN from AddFanStack ;

< Back Finish Cancel Help

Notice that the Maintain language box at the bottom now reads:

`For all include Fannames.CUSTOMER.SSN from AddFanStack;`

15. Click *Finish*.

WebFOCUS Maintain places the source code that the Language Wizard generated in between the Case AddFan and EndCase lines.

Clearing Data From Stacks

Now that you have included the data from AddFanStack into the Fannames data source, it is a good idea to clear the data from AddFanStack. Use the Language Wizard to write this code (see the next section if you get stuck).

Procedure How to Clear the Data From a Stack Using the Language Wizard

1. Place the insertion point after the following statement

```
For all include Fannames.CUSTOMER.SSN from AddFanStack(1);
```

but before

```
EndCase
```

2. Right-click in the Procedure Editor window and, in the shortcut menu, click *Language Wizard*.
3. Select *Operate on a stack* and click *Next*.
4. When the Language Wizard asks you which stack operation you would like to perform, select *Clear the contents of a stack* and click *Next*.
5. The Language Wizard asks you to select one or more stacks to clear. Select *AddFanStack*. Click *Finish*.

The Language Wizard should generate the following code:

```
Stack clear AddFanStack;
```

Assigning the Function to the Add Button

Now that you have written the code that will insert the data the end user enters into the data source, you need to designate that when the end user clicks the Add button, this function gets performed. You do this using the Event Handler editor.

An event is something that an end user performs, such as opening a form, clicking a button, or moving to a field. When you open the Event Handler editor, you will see a list of controls on the form and another list of events that can happen to that control, such as clicking, double-clicking, and so on.

You select a combination of a control and whatever happens to it, and then enter the code for what should happen when this event occurs in the box below.

Procedure How to Assign a Function to an Event

1. In the Form Editor, double-click the *Add* button to open the Event Handler editor.
Notice that AddButton is selected in the list of controls.

2. In the list of events, select *Click*. Notice the code that appears in the Event Handler editor:

```
Case OnAddButton_Click
```

```
EndCase
```

3. Make sure you can see both the Event Handler editor and the Project Explorer.

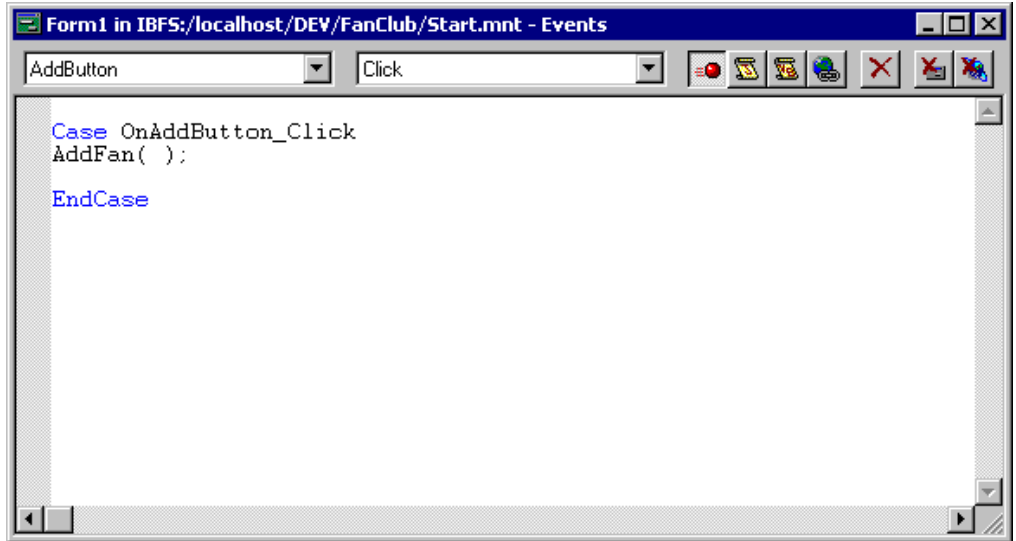
4. Click and drag the *AddFan* function from the Project Explorer into the Event Handler editor between

`Case OnAddButton_Click`

and

`EndCase`

The Event Handler editor will look like the following:



5. Close the Event Handler editor.
6. When WebFOCUS Maintain prompts you to save your procedure, click Yes.
7. Deploy and run your application to see what it looks like.

8. Add your name to the Fannames data source. Welcome to the WebFOCUS Fan Club!

Add a New Member

Ssn: 001-01-0001

First: George Last: Washington

Company: United States

Address: America

City: Washington State: DC Zip: 000001

Phone Ext.: 01

Email: gwashington@whatsemail.com

Mr. (selected)
Mrs.
Ms.

Add

9. Close the application before continuing the tutorial.

Adding a Form to Display Data From a Data Source

The FanClub application can add names to the Fannames data source, but end users do not get much visual feedback from this task. The FanClub application needs to display the contents of the Fannames data source.

This task will be divided into the following steps:

1. Add a new form to the project.
2. Build the code to extract the data from the Fannames data source using the Language Wizard.
3. Design the form to display the contents of Fannames.
4. Create a link from Form1 to the new form.

Procedure How to Add a New Form to Your Project

1. Right-click the *Forms* folder in the Start procedure and, in the shortcut menu, click *New form...*

WebFOCUS Maintain opens the form in the Form Editor.

2. Change the name of the form from Form2 to *ShowFan* by right-clicking Form2 in the Project Explorer and clicking *Rename*.

3. Change the title of the form from Untitled to *Show Fan Club Members*.

(Change the value for the Title property in the property sheet.)

Extracting Data From a Data Source Into a Stack

Your next step is to create a function named *GetFans*, which extracts all the information in the *Fannames* data source and places it into a stack named *GetFanStack*. Try this yourself or see the next section for instructions.

Tip: Create the function and then use the Language Wizard to generate the code.

Procedure How to Extract Data From the Fannames Data Source Into a Stack

1. In the Project Explorer, right-click the *Start* procedure, click *New*, and then click *Function (Case)*

2. In the New Function dialog box, name your function *GetFans* and click *OK*.

3. In the Project Explorer, double-click your new function to open it in the Procedure Editor.

4. Make sure your insertion point is placed after the statement

`Case GetFans`

but before

`EndCase`

5. Open the Language Wizard (right-click in the Procedure Editor window and, from the shortcut menu, click *Language Wizard...*).

6. Select *Retrieve records from a data source* and click *Next*.

7. When WebFOCUS Maintain asks you how you would like to retrieve records from the data source, select *Starting from the current record position... (Next)* and click *Next*.

(The Maintain language contains two commands to retrieve data from a *data source*. This window determines which one you want.)

8. When WebFOCUS Maintain asks you to select the *data source* segments or fields whose records you want to retrieve, in the *Available fields* box, expand the *Fannames data source*, expand the *CUSTOMER* segment, and move the *SSN* field to the *Fields to retrieve* box. Then click *Next*.

(This window is where the Language Wizard determines which *data source* you are reading the data from. Remember that if you select a field in one segment, the rest of the fields in that segment are extracted too. See *Stacks and Implied Columns* on page 2-26 for more information.)

9. When WebFOCUS Maintain asks you how many records you want to retrieve, select *All the records in the selected segment*. Also, make sure *Change the current data source position to the top* is selected, ensuring that WebFOCUS Maintain starts from the beginning of the *data source* when retrieving records. Then click *Next*.
10. When WebFOCUS Maintain asks you to specify a stack, type *GetFanStack* in the text box to create one, and make sure that *Clear the stack first* is selected. You can leave the default value 1 in the *Place the records into the stack...* field. Click *Next*.
11. When WebFOCUS Maintain asks you to supply any conditions, you can leave this window blank because you want to retrieve all records from the data source. Simply click *Finish*.

The Language Wizard should generate the following code:

```
Reposition Fannames.CUSTOMER.SSN;
Stack clear GetFanStack;
For all next Fannames.CUSTOMER.SSN into GetFanStack;
```

Adding an HTML Table to Your Form

You are going to display the fans from the Fannames data source using an HTML table. An HTML table displays the contents of a data source stack in a read-only grid.

(Another option for displaying the fans from the Fannames data source is to create a report procedure in WebFOCUS Developer Studio and execute this code whenever you open this form.)

Procedure How to Add an HTML Table to Your Form

1. Click the *HTML Table* button on the Controls palette.



2. Draw a rectangle on the form representing roughly where you want your HTML table to go on your new ShowFan form.

WebFOCUS Maintain opens the Control Columns dialog box, where you define the contents of your HTML table.

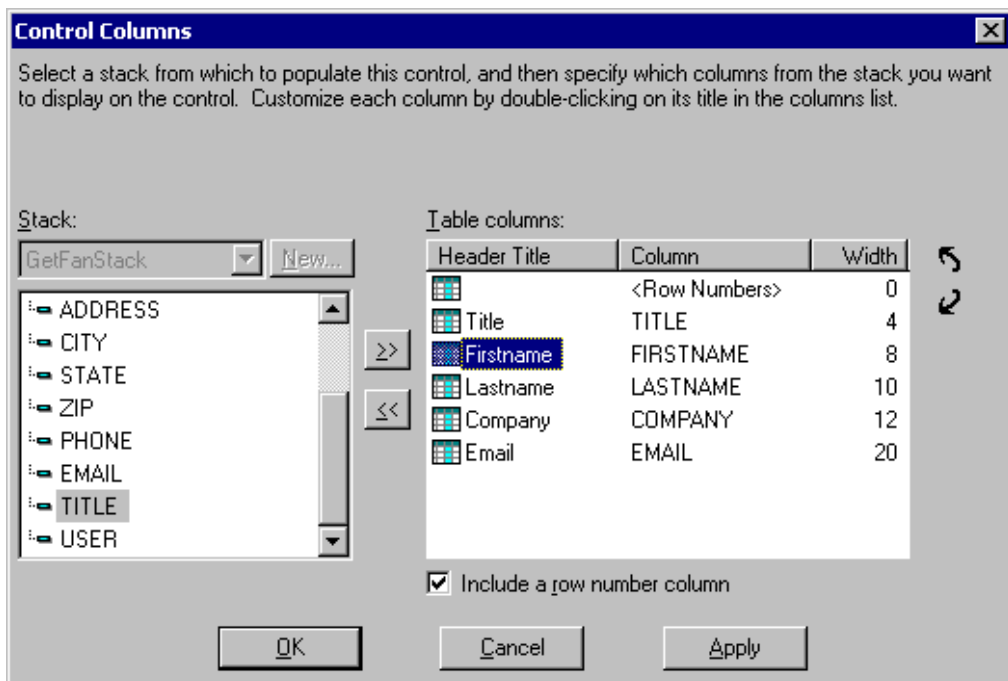
3. Select *GetFanStack* from the Stack list.

Note: Make sure you did not select *AddFanStack*.

The Control Columns dialog box now displays the columns in the stack *GetFanStack*.

4. Copy the fields *LASTNAME*, *FIRSTNAME*, *COMPANY*, *EMAIL*, and *TITLE* into the Table Columns list.

Use the *Move up*  and *Move down*  buttons to rearrange these fields so that they are in the following order: *TITLE*, *FIRSTNAME*, *LASTNAME*, *COMPANY*, and *EMAIL*.



You can change the appearance of any of these columns by opening the Table Column dialog box. For example, suppose you want to change the header titles for the *FIRSTNAME* and *LASTNAME* fields so that they read *First* and *Last*.

5. Double-click *Firstname*.

6. In the Table Column dialog box, change the Header title to *First*.

Table Column

Stack column:

Header title:

Width in characters:

Justification:

Content type:

Links:

Style

Font: ...

Text color:

Background color:

Header style

Font: ...

Text color:

Background color:

You can also change the width, justification, font, and color if you wish.

7. Click *OK*.
8. Repeat the process to change Lastname to *Last*.
9. Click *OK* to leave the HTML Table Columns dialog box.

Your form will resemble the following:

	Title	First	Last	Company	Email
1					
2					
3					
4					
5					

Creating a Link From One Form to Another

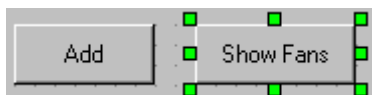
Your final step in creating your new form ShowFan is providing a way to open it from Form1. You will do this by adding a button to Form1 that runs the GetFans function and opens the ShowFan form.

Procedure How to Link From One Form to Another

1. Make Form1 the active window (you can select it from the Window menu or double-click it in the Project Explorer).
2. Click the *Button* control on the Control palette.



3. Draw a rectangle to the right of the Add button on Form1.
4. Change the text in this button to *Show Fans*.
5. Change the name of this button to *ShowFanButton*.

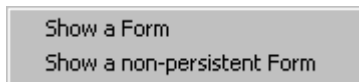


6. Double-click the button to open the Event Handler editor.
7. Select *Click* from the list of events. (ShowFanButton should already be selected in the list of form components.)

WebFOCUS Maintain adds the following code:

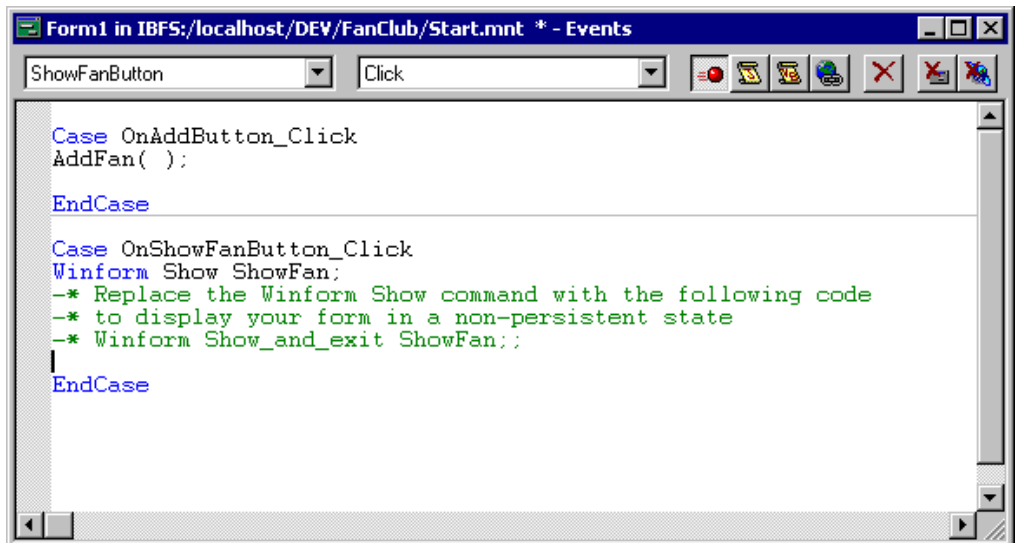
```
Case OnShowFanButton_Click
EndCase
```

8. Drag the *GetFans* function from the Project Explorer into the Event Handler editor between these two lines of code.
9. Drag the *ShowFan* form from the Project Explorer into the Event Handler editor after *GetFans*.
10. You will see the following pop-up window:



Select *Show a Form*.

Comment text appears with the Winform Show command that explains how to show a non-persistent form. Because you want a persistent connection for this form, leave the comment text as is.



11. Close the Event Handler editor.
12. Click Yes to save your procedure.

13. Deploy and run your application to see what it looks like.
14. Close the application before continuing the tutorial.

Using the HTML Overflow Property

When you ran your application, you may have noticed that the HTML table took up enough room to display all of the rows in GetFanStack. If you wish, you can change the HTML table so that it takes up a defined amount of space and has scroll bars so that end users can view rows that do not fit in the space.

This enables you to place information under the HTML table without having to worry about whether the HTML table will overlap it.

Procedure How to Add Scroll Bars to an HTML Table

1. In ShowFan, select the HTML table.
2. In the property sheet, find the Overflow property.
3. Click the value box for the Overflow property.
4. In the list, select 2 - *Scroll*.
5. Deploy and run your application to see what it looks like.

Close the application when you are done before continuing the tutorial.

Adding Form Navigation Buttons

When you ran your application and displayed the Show All Members form, you probably realized that the only thing you could do in this window was close the form with the close box in the upper right corner of the window to go back to the Add a New Member form.

Let's add some navigation buttons to the Fan Club application:

- Add a Back button to the Show Fan Club Members form so that you can go back to the Add a New Member form.
- Add an Exit button to the Add a New Member form.

Procedure How to Add a Back Button

1. Open the *ShowFan* form.
2. Select the *Button* control on the Maintain Controls palette.



3. Draw a button under the HTML table.

4. Change the text on the button to *Back*.
5. Change the name of the button to *BackButton*.
6. Double-click the button to open the Event Handler editor.
7. In the Event Handler editor, select the Click event.
8. Click the *Close form* button on the right side of the window.



This puts the following code in the box:

```
self.WinClose();
```

9. Close the Event Handler editor.
10. Click Yes to save your procedure.

Procedure How to Add an Exit Button

1. Open *Form1*.
2. Place an Exit button named *ExitButton* at the bottom of your form.
Tip: Use the Button control to draw a button at the bottom of your form. Change the text to *Exit* and the name to *ExitButton*.
3. Open the Event Handler editor.
4. In the Event Handler editor, select the Click event.
5. Click the *Close application* button on the right side of the window.



This puts the following code in the box:

```
self.WinExit();
```

6. Close the Event Handler editor.
7. Click Yes to save your procedure.
8. Deploy and run your application to see what it looks like.
9. Click the *Exit* button to close the application before continuing the tutorial.

Adding Images to Your Project

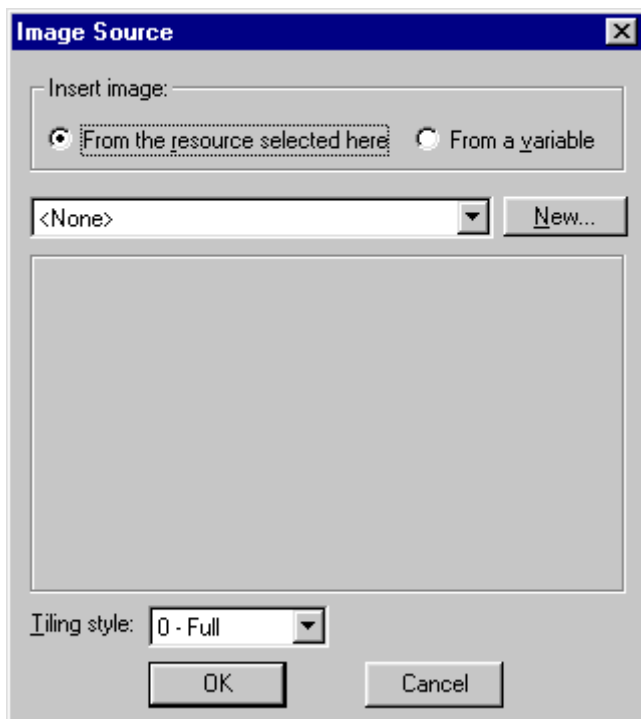
You can use images to improve the appearance and usability of your WebFOCUS Maintain applications. In this section, you will:

- Add an image to the background of your two forms. This image, spiralbg.gif, will make your forms look like pages in a spiral notebook.
- Add titles to your two forms.
- Add a fan graphic to your form.

Procedure How to Add a New Background Image to Your Form


1. Open Form1.
2. Make sure all of the controls are deselected so that you can see the properties for the form.
3. In the property sheet for the form, double-click the *BackgroundImage* property.

WebFOCUS Maintain opens the Image Source dialog box.

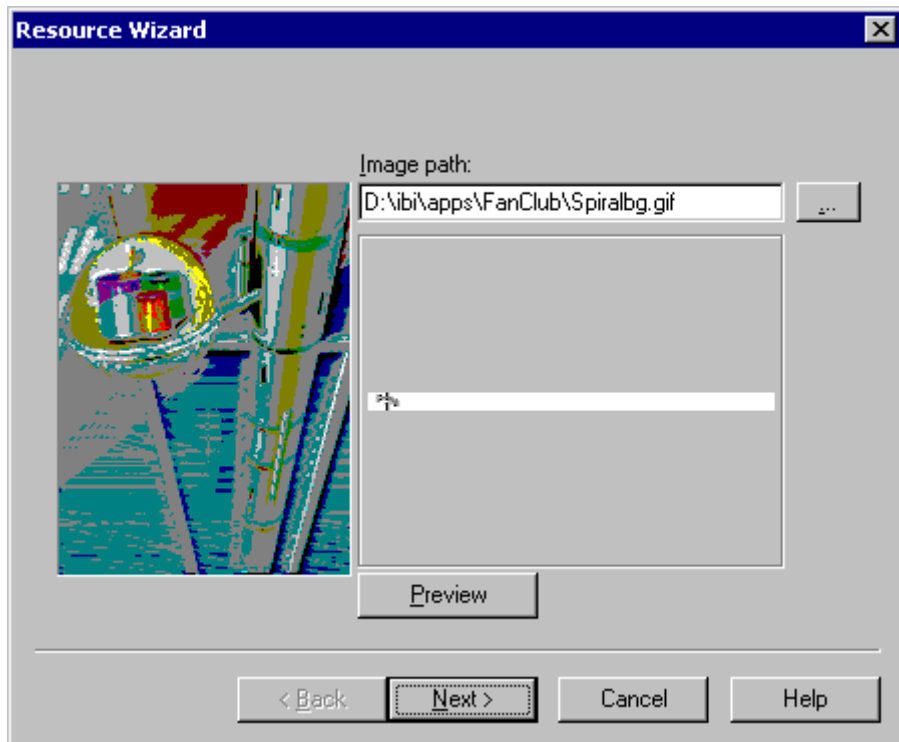


4. Click New.

WebFOCUS Maintain opens the Resource Wizard so that you can define this image as a resource in your project.

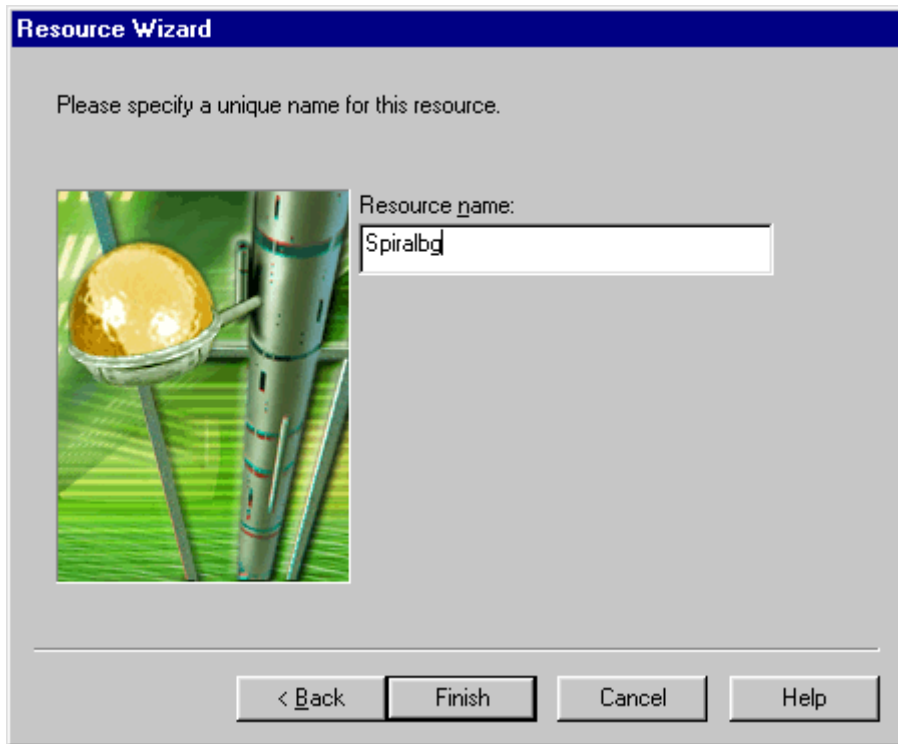
5. When the Resource Wizard asks you to specify the path to the image, click the ellipses  button to use an Open box to find the image, select *spiralbg.gif*, and click the *Open* button.

Note: *Spiralbg.gif* is one of the sample Tutorial files that was placed on your hard drive at installation. It should be located in `\ibi\Apps\Maintain`.



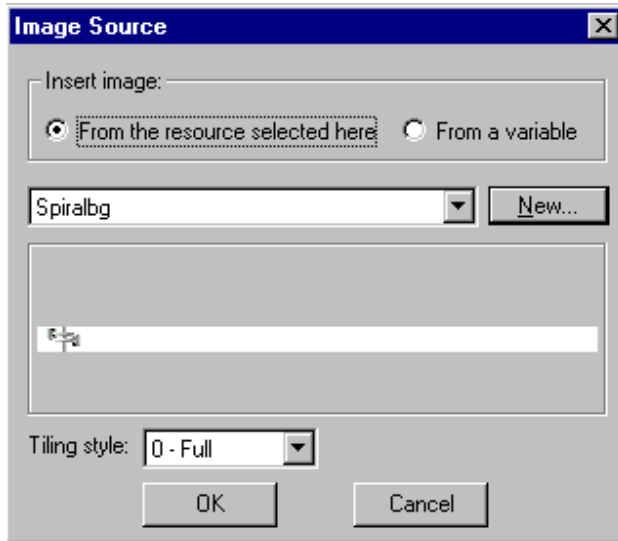
6. Click *Next*.

7. In the final window of the Resource Wizard, you specify a name for your image. By default, WebFOCUS Maintain uses the name of the image.



8. Click *Finish*.
9. WebFOCUS Maintain displays a dialog box informing you that a copy of this file must be created in the project directory. Click *OK*.

You will return to the Image Source dialog box, where your image is now selected as the background image.



10. Click OK.

Your form will resemble the following:

 The screenshot shows a web form titled "Form1 in IBF5:/localhost/DEV/FanClub/Start.mnt". The form is set against a background image of a spiral-bound notebook. It contains several input fields with labels: "Ssn" (value: (AddFanStack.SSN)), "Firstname" (value: (AddFanStack.F)), "Lastname" (value: (AddFanStack.LASTI)), "Company" (value: (AddFanStack.COMPANY)), "Address" (value: (AddFanStack.ADDRESS)), "City" (value: (AddFanStack.CITY)), "State" (value: (A)), "Zip" (value: (AddFanStack)), "Phone" (value: (AddFanStack.PHONE)), and "Email" (value: (AddFanStack.EMAIL)). To the right of the "Ssn" field are three radio buttons labeled "Mr.", "Mrs.", and "Ms.". At the bottom of the form are three buttons: "AddButton", "Show Fans", and "Exit". The form is displayed on a grid with a ruler at the top and left.

You may need to move the controls on your form to the right so that they are not overlapping the spiral. In the Edit menu, click *Select all* and move them.

Tiling Images

If you recall, Spiralbg was a short, wide image, but in your form it seems to be taking up the entire left side. Spiralbg is being tiled, that is, displaying multiple copies of itself. (If you want a repetitive design on your form, it is better to use a small image and tile it instead of using a large image with the full picture you want. This conserves memory and ensures that the image will look good no matter how big or small an end user's window is.)

The Image Source dialog box automatically tiled Spiralbg by selecting *0 - Full* in the Tiling style list.

Using Folders in the Explorer

WebFOCUS Maintain has now added this graphic to your project.

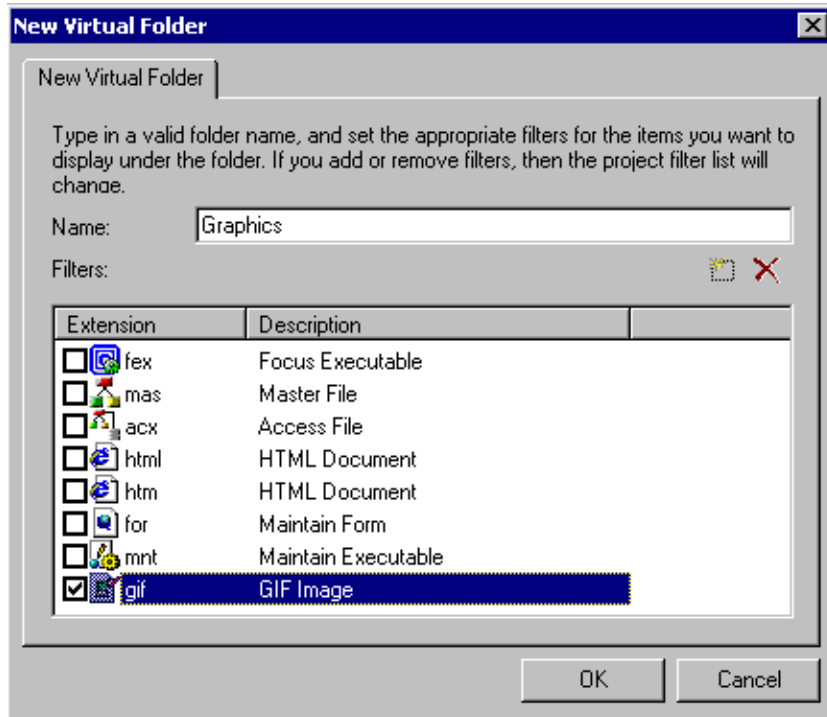
To view the graphic, you need to display the graphics folder.

This means that WebFOCUS Maintain now knows where to find this image and you can easily add it to other forms in your project. Let's add it to ShowFan.

***Procedure* How to View Graphics Folders**

1. In the Explorer, right-click FanClub, click *New*, and then click *Virtual Folder* in the submenu.
2. In the New Virtual Folder dialog box, enter *Graphics* in the Name box.

3. Check off *GIF Image* as the contents of this folder.



4. Click *OK*.

WebFOCUS Maintain adds the Graphics folder to the folders under FanClub. If you open it, you will see SpiralBg.

Procedure How to Add an Existing Background Image to Your Form

1. Open the ShowFan form.
2. Make sure all objects on the form are deselected so that you can see the properties for the form.
3. In the property sheet for the form, double-click the *BackgroundImage* property. WebFOCUS Maintain opens the Image Source dialog box.
4. Select *Spiralbg* from the list of available images.
5. Click *OK*.
6. If necessary, move the HTML table to the right.
7. Deploy and run your application to see what it looks like.
8. Click the *Exit* button to close the application before continuing the tutorial.

Procedure How to Add an Image to Your Form

1. Open Form1.
2. Select all the controls on your form by pressing **CTRL+A** and move them down so that you have roughly one inch at the top of your form.

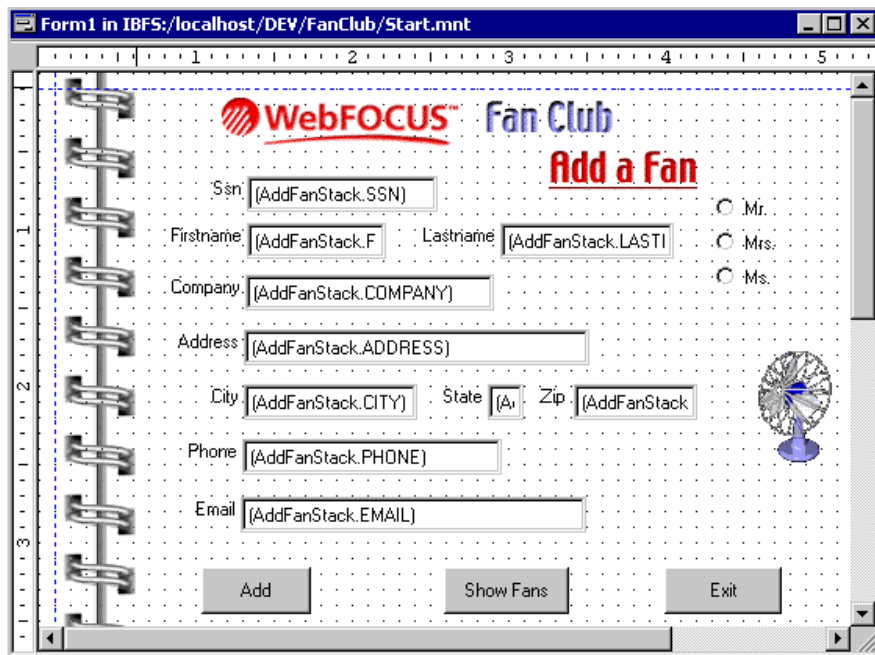
Note: Use your rulers to determine how much room to leave at the top. If your rulers are not visible, select *Rulers* from the View menu or click the *Toggle rulers* button on the Layout toolbar.

3. Select the *Image* control button.



4. Draw a box in the empty space at the top of the form.
WebFOCUS Maintain opens the Image Source dialog box.
5. Repeat the steps you followed in *How to Add a New Background Image to Your Form* on page 2-46 to add Addafan.gif to your form.
6. Add the *Fan.gif* image to your form to the right of the entry boxes. (Repeat steps 3 through 5.)

Your form will resemble the following:



7. Open ShowFan and add the image *Currfan.gif*. (Repeat steps 3 through 5.)
8. Deploy and run your application to see what it looks like.

What's Next?

To learn more about WebFOCUS Maintain, try Chapter 3, *The WebFOCUS Maintain Advanced Tutorial*.

What's Next?

CHAPTER 3

The WebFOCUS Maintain Advanced Tutorial

This chapter is under development and will be released at a later date.

CHAPTER 4

WebFOCUS Maintain Concepts

Topics:

- Set-based Processing
- Controlling a Procedure's Flow
- Executing Other Maintain Procedures
- Executing External Procedures
- Forms and Event-driven Processing
- Reading From a Data Source
- Writing to a Data Source
- Transaction Processing
- Classes and Objects

The following topics provide an overview of basic WebFOCUS Maintain concepts. To fully exploit the potential and productivity of WebFOCUS Maintain, you should become familiar with concepts including:

- Processing data in sets by using stacks.
- Controlling the flow of a WebFOCUS Maintain project and the procedures within it.
- Developing presentation logic using forms and event handlers.
- Reading from and writing to data sources.
- Ensuring transaction integrity.
- Creating classes and objects.

Set-based Processing

Maintain provides the power of set-based processing, enabling you to read, manipulate, and write groups of records at a time. You manipulate these sets of data using a data structure called a *data source stack*.

A data source stack is a simple temporary table. Generally, columns in a data source stack correspond to data source fields, and rows correspond to records, or path instances, in that data source. You can also create your own “user-defined” columns.

The intersection of a row and a column is called a cell and corresponds to an individual field value. The data source stack itself represents a data source path.

For example, consider the following Maintain command:

```
FOR ALL NEXT Emp_ID Pay_Date Ded_Amt INTO PayStack
WHERE Employee.Emp_ID EQ SelectedEmpID;
```

This command retrieves Emp_ID and the other root segment fields, as well as the Pay_Date, Gross, Ded_Code, and Ded_Amt fields from the Employee data source and holds them in a data source stack named PayStack. Because the command specifies FOR ALL, it retrieves all of the records at the same time; you do not need to repeat the command in a loop. Because it specifies WHERE, it retrieves only the records you need—in this case, the payment records for the currently-selected employee.

You could just as easily limit the retrieval to a sequence of data source records, such as the first 6 payment records that satisfy your selection condition

```
FOR 6 NEXT Emp_ID Pay_Date Ded_Amt INTO PayStack
WHERE Employee.Emp_ID EQ SelectedEmpID;
```

or even restrict the retrieval to employees in the MIS department earning salaries above a certain amount:

```
FOR ALL NEXT Emp_ID Pay_Date Ded_Amt INTO PayStack
WHERE (Employee.Department EQ 'MIS') AND
      (Employee.Curr_Sal GT 23000);
```

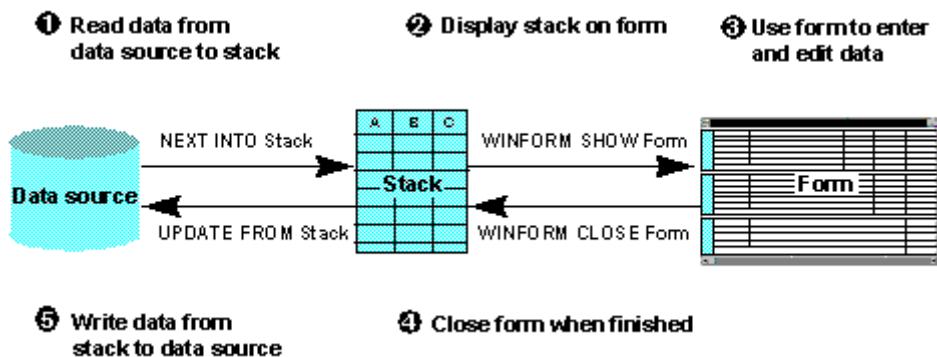
Which Processes Are Set-based?

You can use set-based processing for the following types of operations:

- **Selecting records.** You can select a group of data source records at one time using the NEXT command with the FOR prefix. Maintain retrieves all of the data source's records that satisfy the conditions you specified in the command and then automatically puts them into the data source stack that you specified.

- **Collecting transaction values.** You can use forms to display, edit, and enter values for groups of rows. The rows are retrieved from a data source stack, displayed in the form, and are placed back into a stack when the user is finished. You can also use the NEXT command to read values from a transaction file into a stack.
- **Writing transactions to the data source.** You can include, update, or delete a group of records at one time using the INCLUDE, UPDATE, REVISE, or DELETE commands with the FOR prefix. The records come from the data source stack that you specify in the command.
- **Manipulating stacks.** You can copy a set of records from one data source stack to another and sort the records within a stack.

The following diagram illustrates how these operations function together in a procedure:



The diagram is explained in detail below:

1. The procedure selects several records from the data source and, for each record, copies the values for fields A, B, and C into the data source stack. It accomplishes this using the NEXT command.
2. The procedure displays a form on the screen. The form shows multiple instances of fields A, B, and C; the field values shown on the screen are taken from the stack. This is accomplished using a form and the WINFORM SHOW command.
3. The procedure user views the form and enters and edits data. As the form responds to the user's activity, it automatically communicates with the procedure and updates the stack with the new data.
4. The procedure user clicks a button to exit the form; the button accomplishes this by triggering the WINFORM CLOSE command.
5. The procedure writes the values for fields A, B, and C from the stack to the selected records in the data source. The procedure accomplishes this using the UPDATE command.

How Does Maintain Process Data in Sets?

Maintain processes data in sets using two features:

- **The command prefix FOR.** When you specify FOR at the beginning of the NEXT, INCLUDE, UPDATE, REVISE, and DELETE commands, the command works on a group of records, instead of on just one record.
- **Stacks.** You use a data source stack to hold the data from a group of data source or transaction records. For example, a stack can hold the set of records that are output from one command (such as NEXT or WINFORM) and provide them as input to another command (such as UPDATE). This enables you to manipulate the data as a group.

Creating and Defining Data Source Stacks: An Overview

Maintain makes working with stacks easy by enabling you to create and define a data source stack dynamically, simply by using it. For example, when you specify a particular stack as the destination stack for a data source retrieval operation, that stack is defined as including all of the fields in all of the segments referred to by the command. Consider the following NEXT command, which retrieves data from the VideoTrk data source into the stack named VideoTapeStack:

```
FOR ALL NEXT CustID INTO VideoTapeStack;
```

Because the command refers to the CustID field in the Cust segment, all of the fields in the Cust segment (from CustID through Zip) are included as columns in the stack. Every record retrieved from the data source is written as a row in the stack.

Example Creating and Populating a Simple Data Source Stack

If you are working with the VideoTrk data source, and you want to create a data source stack containing the ID and name of all customers whose membership expired after June 24, 1992, you could issue the following NEXT command:

```
FOR ALL NEXT CustID INTO CustNames WHERE ExpDate GT 920624;
```

The command does the following:

1. Selects (NEXT) all VideoTrk records (FOR ALL) that satisfy the membership condition (WHERE).
2. Copies all of the fields from the Cust segment (referenced by the CustID field) from the selected data source records into the CustNames stack (INTO).

The resulting CustNames stack looks like this (some intervening columns have been omitted to save space):

CustID	LastName	...	Zip
0925	CRUZ	...	61601
1118	WILSON	...	61601
1423	MONROE	...	61601
2282	MONROE	...	61601
4862	SPIVEY	...	61601
8771	GARCIA	...	61601
8783	GREEN	...	61601
9022	CHANG	...	61601

Creating a Data Source Stack

You create a data source stack:

- **Implicitly**, by specifying it in a NEXT or MATCH command as the destination (INTO) stack, or by associating it in the Form Editor with a control.

Forms are introduced in *Forms and Event-driven Processing* on page 4-23; the Form Editor used to design and create forms is described in *Using the Form Editor* in *Using WebFOCUS Maintain*.

- **Explicitly**, by declaring it in an INFER command.

For example, this NEXT command creates the EmpAddress stack:

```
FOR ALL NEXT StreetNo INTO EmpAddress;
```

Defining a Data Source Stack's Data Source Columns

When you define a data source stack, you can include any field along a data source path. Maintain defines a stack's data source columns by performing the following steps:

1. Scanning the procedure to identify all the NEXT, MATCH, and INFER commands that use the stack as a destination and all the controls that use the stack as a source or destination.

2. Identifying the data source fields that these commands and controls move in or out of the stack:
 - **NEXT commands** move the fields in the data source field list and WHERE phrase.
 - **MATCH commands** move the fields in the data source field list.
 - **INFER commands** move all the fields specified by the command.
 - **Controls** move all the fields specified by the control.
3. Identifying the data source path that contains these fields.
4. Defining the stack to include columns corresponding to all the fields in this path.

You can include any number of segments in a stack, as long as they all come from the same path. When determining a path, unique segments are interpreted as part of the parent segment. The path can extend through several data sources that have been joined together. Maintain supports joins that are defined in a Master File. For information about defining joins in a Master File, see *Describing Data With WebFOCUS Language*. (Maintain can read from joined data sources, but cannot write to them.)

The highest specified segment is known as the anchor and the lowest specified segment is known as the target. Maintain creates the stack with all of the segments needed to trace the path from the root segment to the target segment:

- It automatically includes all fields from all of the segments in the path that begins with the anchor and continues to the target.
- If the anchor is not the root segment, it automatically includes the key fields from the anchor's ancestor segments.

Example Defining Data Source Columns in a Data Source Stack

In the following source code, a NEXT command refers to a field (Last_Name) in the EmplInfo segment of the Employee data source, and reads that data into EmpStack; another NEXT command refers to a field (Salary) in the PayInfo segment of Employee and also reads that data into EmpStack:

```
NEXT Last_Name INTO EmpStack;  
.  
.  
.  
FOR ALL NEXT Salary INTO EmpStack;
```

Based on these two NEXT commands, Maintain defines a stack named EmpStack, and defines it as having columns corresponding to all of the fields in the EmplInfo and PayInfo segments:

Emp_ID	Last_Name	...	Ed_Hrs	Dat_Inc	...	Salary	JobCode
071382660	STEVENS	...	25.00	82/01/01	...	\$11,000.00	A07
071382660	STEVENS	...	25.00	81/01/01	...	\$10,000.00	A07

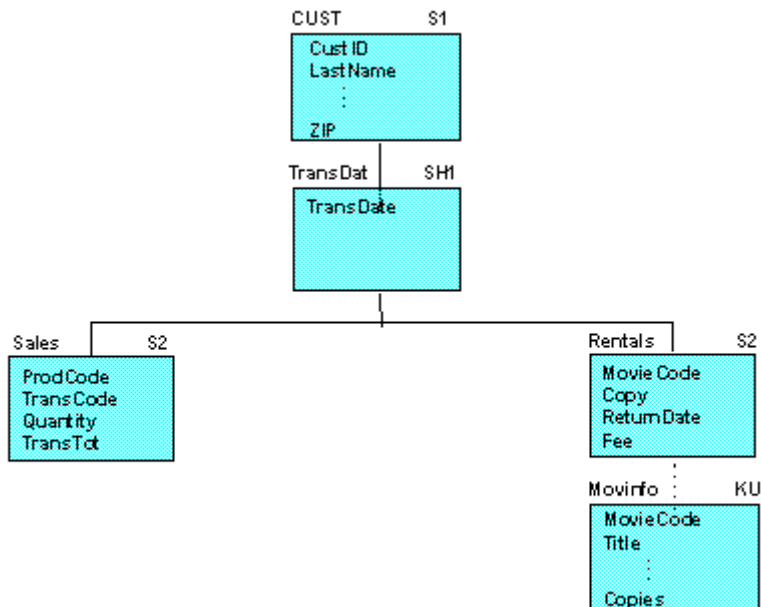
Example Establishing a Path Using Keys and Anchor and Target Segments

The following code populates CustMovies, a data source stack that contains video rental information for a given customer. The first NEXT command identifies the customer. The second NEXT command selects a field (TransDate) from the second segment and a field (Title) from the bottom segment of a path that runs through the joined VideoTrk and Movies data sources:

```
NEXT CustID WHERE CustID IS '7173';
```

```
FOR ALL NEXT TransDate Title INTO CustMovies
  WHERE Category IS 'COMEDY';
```

The structure of the joined VideoTrk and Movies data sources looks like this:



In this NEXT command, the TransDat segment is the anchor and the MovInfo segment is the target. The resulting CustMovies stack contains all the fields needed to define the data source path from the root segment to the target segment:

- The anchor's ancestor segment, Cust (key field only).
- All segments from the anchor through the root: TransDat, Rentals, MovInfo (all fields).

The stack looks like this:

CustID	TransDate	MovieCode	...	Title	...	Copies
7173	91/06/18	305PAR	...	AIRPLANE	...	2
7173	91/06/30	651PAR	...	MY LIFE AS A DOG	...	3

Creating a Data Source Stack's User-defined Columns

In addition to creating data source stack columns that correspond to data source fields, you can also create data source stack columns that you define yourself. You can define these columns in two ways:

- **Within a procedure.** You can create a stack column (as well as a user-defined variable) by issuing a COMPUTE command. You can also use the COMPUTE command to assign values to stack cells.

Because all Maintain variables are local to a procedure, you must redefine variables in each procedure in which you use them. For user-defined stack columns, you accomplish this by simply reissuing the original COMPUTE command in each procedure to which you are passing the stack. (You only need to specify the variable's format; do not specify its value, which is passed with the stack.)

- **Within the Master File.** You can define a virtual field in a Master File by using the DEFINE attribute. The field is then available in every procedure that accesses the data source. The virtual field is treated as part of the data source segment in which it is defined, and Maintain automatically creates a corresponding column for it—a virtual column—in every stack that references that segment.

Virtual fields must be derived, directly or indirectly, from data source values. They cannot be defined as a constant. The expression assigned to a virtual field in the Master File can reference fields from other segments in the same data source path as the virtual field.

Unlike other kinds of stack columns, you cannot update a virtual column or field, and you cannot test it in a WHERE phrase.

Example Creating a User-defined Column

Consider a data source stack named Pay that contains information from the Employee data source. If you want to create a user-defined column named Bonus and set its value to 10% of each employee's current salary, you could issue the COMPUTE command to create the new column, and then issue another COMPUTE to derive the value. You place the second COMPUTE within a REPEAT loop to execute it once for each row in the stack:

```
COMPUTE Pay.Bonus/D10.2;
REPEAT Pay.FocCount Row/I4=1;
    COMPUTE Pay(Row).Bonus = Pay(Row).Curr_Sal * .10;
ENDREPEAT Row=Row+1;
```

Copying Data Into and Out of a Data Source Stack

You can copy data into and out of a data source stack in the following ways:

- **Between a stack and a data source.** You can copy data from a data source into a stack using the NEXT and MATCH commands. You can copy data in the opposite direction, from a stack into a data source, using the INCLUDE, UPDATE, and REVISE commands. In addition, the DELETE command, while not actually copying a stack's data, reads a stack to determine which records to remove from a data source. For more information about these commands, see *Command Reference* in *Language Reference*.
- **Between a stack and a form.** You can copy data from a stack into a form, and from a form into a stack, by specifying the stack as the source or destination of the data displayed by the form. This technique makes it easy for an application user to enter and edit stack data at a personal computer. For more information about using stacks with forms, see *Developing and Using Controls* in *Developing WebFOCUS Maintain Applications*.
- **From a transaction file to a stack.** You can copy data from a transaction file to a stack using the NEXT command. For more information about this command, see *Command Reference* in *Language Reference*.
- **Between two stacks.** You can copy data from one stack to another using the COPY and COMPUTE commands. For more information about these commands, see *Command Reference* in *Language Reference*.

You can use any of these commands to copy data by employing the command's INTO and FROM phrases. FROM specifies the command's data source (the source stack), and INTO specifies the command's data destination (the destination stack).

Example Copying Data Between a Data Source Stack and a Data Source

In this NEXT command

```
FOR ALL NEXT CustID INTO CustStack;
```

the INTO phrase copies the data (the CustID field and all of the other fields in that segment) into CustStack. The following UPDATE command

```
FOR ALL UPDATE ExpDate FROM CustStack;
```

uses the data from CustStack to update records in the data source.

Referring to Specific Stack Rows Using an Index

Each stack has an index that enables you to refer to specific rows. For example, by issuing a NEXT command, we create the CustNames stack to retrieve records from the VideoTrk data source:

```
FOR ALL NEXT CustID LastName INTO CustNames
WHERE ExpDate GT 920624;
```

The first record retrieved from VideoTrk becomes the first row in the data source stack, the second record becomes the second row, and so on.

	CustID	LastName	...	Zip
1	0925	CRUZ	...	61601
2	1118	WILSON	...	61601
3	1423	MONROE	...	61601
4	2282	MONROE	...	61601
5	4862	SPIVEY	...	61601
6	8771	GARCIA	...	61601
7	8783	GREEN	...	61601
8	9022	CHANG	...	61601

You can refer to a row in the stack by using a subscript. The following example refers to the third row, in which CustID is 1423:

```
CustNames (3)
```

You can use any integer value as a subscript: an integer literal (such as 3), an integer field (such as TransCode), or an expression that resolves to an integer (such as TransCode + 2).

You can even refer to a specific column in a row (that is, to a specific stack cell) by using the stack name as a qualifier:

```
CustNames(3).LastName
```

If you omit the row subscript, the position defaults to the first row. For example,

```
CustNames.LastName
```

is equivalent to

```
CustNames(1).LastName
```

Maintain provides two system variables associated with each stack. These variables help you to navigate through a stack and to manipulate single rows and ranges of rows:

- **FocCount.** This variable's value is always the number of rows currently in the stack and is set automatically by Maintain. This is very helpful when you loop through a stack, as described in the following section, *Looping Through a Stack* on page 4-11. FocCount is also helpful for checking if a stack is empty:

```
IF CustNames.FocCount EQ 0 THEN PERFORM NoData;
```

- **FocIndex.** This variable points to the current row of the stack. When a stack is displayed in a form, the form sets FocIndex to the row currently selected in the grid or browser. Outside of a form, the developer sets the value of FocIndex. By changing its value, you can point to any row you wish. For example, in one function you can increment FocIndex for the Rental stack:

```
IF Rental.FocIndex LT Rental.FocCount
  THEN COMPUTE Rental.FocIndex = Rental.FocIndex + 1;
```

You can then invoke a second function that uses FocIndex to retrieve desired records into the MovieList stack:

```
FOR ALL NEXT CustID MovieCode INTO MovieList
  WHERE VideoTrk.CustID EQ Rental(Rental.FocIndex).CustID;
```

The syntax "*stackname(stackname.FocIndex)*" is identical to "*stackname()*", so you can code the previous WHERE phrase more simply as follows:

```
WHERE VideoTrk.CustID EQ Rental().CustID
```

Looping Through a Stack

The REPEAT command enables you to loop through a stack. You can control the process in different ways, so that you can loop according to several factors:

- The number of times specified by a literal, or by the value of a field or expression.
- The number of rows in a stack, by specifying the stack's FocCount variable.
- While an expression is true.

- Until an expression is true.
- Until the logic within the loop determines that the loop should be exited.

You can also increment counters as part of the loop.

Example Using REPEAT to Loop Through a Stack

The following REPEAT command loops through the Pay stack once for each row in the stack and increments the temporary variable Row by one for each loop:

```
REPEAT Pay.FocCount Row/I4=1;  
    COMPUTE Pay(Row).NewSal = Pay(Row).Curr_Sal * 1.10;  
ENDREPEAT Row=Row+1;
```

Sorting a Stack

You can sort a stack's rows using the STACK SORT command. You can sort the stack by one or more of its columns and sort each column in ascending or descending order. For example, the following STACK SORT command sorts the CustNames stack by the LastName column in ascending order (the default order): STACK SORT CustNames BY LastName;

Editing and Viewing Stack Values

There are multiple ways in which you can edit and/or view a stack's values:

- **Forms.** You can display a stack in an HTML table or a grid on a form. A grid allows you to edit a stack's fields directly on the screen. You cannot edit a stack in an HTML table.
- **COMPUTE command.** You can use the COMPUTE command to assign a value to any of a stack's cells. (When assigning a value, the COMPUTE keyword is optional, as described in *Command Reference in Language Reference*.) For example, the following command assigns the value 35000 to the cell at the intersection of row 7 and column NewSal in the Pay stack:

```
COMPUTE Pay(7).NewSal = 35000;
```

It is important to note that if you do not specify a row when you assign values to a stack, Maintain defaults to the first row. Thus if the Pay stack has 15 rows, and you issue the following command

```
COMPUTE Pay.NewSal = 28000;
```

the first row receives the value 28000. If you issue this NEXT command

```
FOR 6 NEXT NewSal INTO Pay;
```

the current row of Pay defaults to one, and so the six new values are written to rows one through six of Pay; any values originally in the first six rows of Pay will be overwritten.

If you wish to append the new values to Pay—that is, to add them as new rows 16 through 21—you would issue this NEXT command, which specifies the starting row:

```
FOR 6 NEXT NewSal INTO Pay(16);
```

You can accomplish the same thing without needing to know the number of the stack's last row by using the stack's FocCount variable:

```
FOR 6 NEXT NewSal INTO Pay(Pay.FocCount+1);
```

If you want to discard the original contents of Pay and substitute the new values, it is best to clear the stack before writing to it using the following command:

```
STACK CLEAR Pay;
FOR 6 NEXT NewSal INTO Pay;
```

The Default Data Source Stack: The Current Area

For all data source fields referenced by a Maintain procedure, Maintain creates a corresponding column in the default data source stack known as the *Current Area*.

The Current Area is always present and is global to the procedure. It has one row, and functions as a kind of data source buffer; each data source field—that is, each field described in a Master File that is accessed by a Maintain procedure—has a corresponding column in the Current Area. When a data source command assigns a value—either to a field via INCLUDE, UPDATE, or REVISE, or from a field to a stack via NEXT or MATCH—Maintain automatically assigns that same value to the corresponding column in the single row of the Current Area. If a set-based data source command writes multiple values to or from a stack column, the last value that the command writes is the one that is retained in the Current Area.

Tip: Stacks are a superior way of manipulating data source values: because the Current Area is a buffer, it does not function as intuitively as stacks do. It is recommended that you use stacks instead of the Current Area to manipulate data source values.

For example, if you write 15 values of NewSal to the Pay stack, the values will also be written to the NewSal column in the Current Area; since the Current Area has only one row, its value will be the fifteenth (that is, the last) value written to the Pay stack.

The Current Area is the default stack for all FROM and INTO phrases in Maintain commands. If you do not specify a FROM stack, the values come from the single row in the Current Area; if you do not specify an INTO stack, the values are written to the single row of the Current Area, so that only the last value written remains.

The standard way of referring to a stack column is by qualifying it with the stack name and a period:

```
stackname.columnname
```

Because the Current Area is the default stack, you can explicitly reference its columns without the stack name, by prefixing the column name with a period:

.columnname

Within the context of a WHERE phrase, an unqualified name refers to a data source field (in a NEXT command) or a stack column (in a COPY command). To refer to a Current Area column in a WHERE phrase you should reference it explicitly by qualifying it with a period. Outside of a WHERE phrase it is not necessary to prefix the name of a Current Area column with a period, as unqualified field names will default to the corresponding column in the Current Area.

For example, the following NEXT command compares Emp_ID values taken from the Employee data source with the Emp_ID value in the Current Area:

```
FOR ALL NEXT Emp_ID Pay_Date Ded_Code INTO PayStack
WHERE Employee.Emp_ID EQ .Emp_ID;
```

If the Current Area contains columns for fields with the same field name but located in different segments or data sources, you can distinguish between the columns by qualifying each one with the name of the Master File and/or segment in which the field is located:

masterfile_name.segment_name.column_name

If a user-defined variable and a data source field have the same name, you can qualify the name of the data source field's Current Area column with its Master File and/or segment name; an unqualified reference will refer to the user-defined variable.

Maximizing Data Source Stack Performance

When you use data source stacks, there are several things you can do to optimize performance:

- **Filter out unnecessary rows.** When you read records into a stack, you can prevent the stack from growing unnecessarily large by using the WHERE phrase to filter out unneeded rows.
- **Clear stacks when done with data.** Maintain automatically releases a stack's memory at the end of a procedure, but if in the middle of a procedure you no longer need the data stored in a stack, you can clear it immediately by issuing the STACK CLEAR command. Clearing the data frees the stack's memory for use elsewhere.
- **Do not reuse a stack for an unrelated purpose.** When you specify a stack as a data source or destination in certain contexts (in the NEXT, MATCH, and INFER commands, and in the Form Editor for controls), you define the columns that the stack will contain. If you use the same stack for two unrelated purposes, it will be created with the columns needed for both, making it unnecessarily wide.

Controlling a Procedure's Flow

Maintain provides many different ways of controlling the flow of execution within a procedure. You can:

- **Nest** a block of code. In commands in which you can nest another command—such as in an IF command—you can nest an entire block of commands in place of a single one by defining the block using the BEGIN command. BEGIN is described in *Language Reference*.
- **Loop** through a block of code a set number of times, while a condition remains true, or until it becomes true, using the REPEAT command. REPEAT is described in *Language Reference*.
- **Branch unconditionally** to a block of code called a Maintain function. You define the function using the CASE command, and can invoke it in a variety of ways. When the function terminates, it returns control to the command following function invocation. The CASE command is described in *Language Reference*.

Alternatively, you can branch to a function, but not return upon termination, by invoking the function using the GOTO command. GOTO is described in *Language Reference*.

- **Branch conditionally** using the IF command. If the expression you specify in the IF command is true, the command executes a PERFORM or GOTO command nested in the THEN phrase, which branches to a Maintain function. IF is described in *Language Reference*.

Alternatively, you can nest a different command, such as a BEGIN command defining a block of code, to be conditionally executed by the IF command.

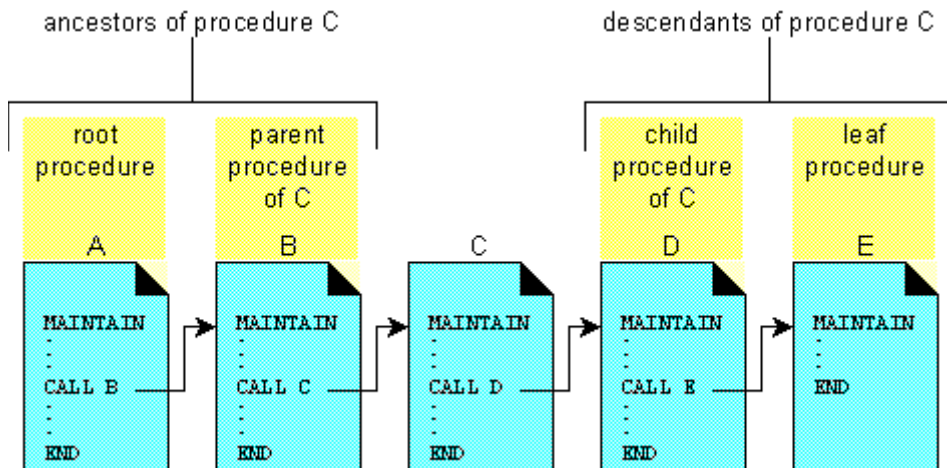
- **Trigger** an event handler in response to a user event. When a user performs the event in a form at run time, the event triggers the event handler—a function or URL link—that you have specified. Event handlers are described in *Forms and Event-driven Processing* on page 4-23, and in *Defining Events and Event Handlers* in *Developing WebFOCUS Maintain Applications*.

Executing Other Maintain Procedures

You can call one Maintain procedure from another with the CALL command. (*Maintain procedure* here means any procedure of Maintain language commands.) CALL simplifies the process of modularizing an application; software designed as a group of related modules tends to be easier to debug, easier to maintain, and lends itself to being reused by other applications, all of which increase your productivity.

CALL also makes it easy to partition an application, deploying each type of logic on the platform on which it will run most effectively.

The following diagram illustrates how to describe the relationship between called and calling procedures. It describes a sequence of five procedures from the perspective of the middle procedure, which is named C. (Note that a root procedure is also called the starting procedure.)



Calling a Maintain Procedure on a Different Server

If parent and child Maintain procedures reside on different servers, you identify the location of the child procedure in a deployment scenario in the Maintain Development Environment. At deployment time, the Maintain Development Environment automatically informs the parent procedure of the location of the child procedure by supplying the *AT server* phrase in the *CALL* command. Do not code the *AT server* phrase yourself unless the parent procedure is a linked procedure.

If the parent procedure is linked to the project instead of being a native project component, code the *AT server* phrase yourself in the parent procedure's *CALL* command (or use the Language Wizard to generate the *CALL* command) to identify the location of the child procedure.

Example **Calling the EmpUpdat Procedure on a Different Server**

Consider the EmpUpdat procedure:

```
MAINTAIN FILE Employee
FOR ALL NEXT Emp_ID INTO EmpStack;
.
.
.
CALL NewClass;
.
.
.
END
```

This calls the NewClass procedure that is deployed on the EducServ WebFOCUS Server:

```
MAINTAIN
.
.
.
END
```

In this example, EmpUpdat is the parent procedure and NewClass is the child procedure. When the child procedure—and any procedures that it has invoked—have finished executing, control returns to the parent.

Passing Variables Between Procedures

All user variables (both stacks and simple, or scalar, variables) are global to a function or procedure, but not global to the project. In other words, to protect them from unintended changes in other parts of a project, you cannot directly reference a variable outside of the procedure in which it is found (with the exception of the FocError transaction variable). However, you can access a variable's data in other procedures, simply by passing it as an argument from one procedure to another.

To pass variables as arguments, you only need to name them in the CALL command, and then again in the corresponding MAINTAIN command, using the commands' FROM phrase for input arguments and INTO phrase for output arguments. Some variable attributes must match in the CALL and MAINTAIN commands:

- **Number.** The number of arguments in the parent and child procedures must be identical.
- **Sequence.** The order in which you name stacks and simple variables must be identical in the CALL and corresponding MAINTAIN commands.
- **Data type.** Stack columns and simple variables must have the same data type (for example, integer) in both the parent and child procedures.

- **Stack column names.** The names of stack columns need to match; if a column has different names in the parent and child procedures, it is not passed.

Other attributes need not match:

- **Stack and scalar variable names.** The names of stacks and simple variables specified in the two commands need not match.
- **Other data attributes.** All other data attributes, such as length and precision, do not need to match.
- **Simple variables.** If you pass an individual stack cell, you must receive it as a simple variable, not as a stack cell.

Once you have passed a variable to a child procedure, you need to define it in that procedure. How you define it depends upon the type of variable:

- **User-defined columns and fields.** You must redefine each user-defined variable using a DECLARE or COMPUTE command. You only need to specify the variable's format, not its value. For example, the following DECLARE command redefines the Counter field and the FullName column:

```
DECLARE Counter/A20;  
EmpStack.FullName/A15;
```

- **Data source and virtual stack columns.** You can define a stack's data source columns and virtual columns in one of two ways: implicitly, by referring to the stack columns in a data source command, or explicitly, by referring to them using the INFER command. For example:

```
INFER Emp_ID Pay_Date INTO EmpStack;
```

The INFER command declares data source fields and the stack with which they are associated. You can specify one field for each segment you want in the stack or simply one field each from the anchor and target segments of a path you want in the stack.

While INFER reestablishes the stack's definition, it does not retrieve any records from the data source.

Once a variable has been defined in the child procedure, its data becomes available. If you refer to stack cells that were not assigned values in the parent procedure, they are assigned default values (such as spaces or zeros) in the child procedure, and a message is displayed warning that they have not been explicitly assigned any values.

When the child procedure returns control back to the parent procedure, the values of stacks and simple variables specified as output arguments are passed back to the parent. The values of stacks and simple variables specified only as input arguments are not passed back.

Example Passing Data Between Maintain Procedures

This procedure

```
MAINTAIN FILE Employee
FOR ALL NEXT Emp_ID INTO EmpStack;
.
.
.
CALL NewClass FROM EmpStack CourseStack INTO CourseStack;
.
.
.
END
```

calls the NEWCLASS procedure:

```
MAINTAIN FROM StudentStack CourseStack INTO CourseStack
.
.
.
END
```

EmpStack and CourseStack in the parent procedure correspond to StudentStack and CourseStack in the child procedure.

Accessing Data Sources in the Child Procedure

If a child procedure accesses a data source, whether retrieving or writing records, you must specify the data source in the MAINTAIN command. This is done the same way as for a stand-alone procedure. For example, the procedure below specifies the Employee and EducFile data sources:

```
MAINTAIN FILES Employee AND EducFile FROM StuStk INTO CoursStk
.
.
.
END
```

Data Source Position in Child Procedures

Each Maintain procedure tracks its own position in the data source. When you first call a procedure, Maintain positions you at the beginning of each segment in each data source accessed within that procedure; after navigating through a data source you can reposition to the beginning of a segment by issuing the REPOSITION command. Each procedure's data source positions are independent of the positions established in other procedures.

When a child procedure returns control to its parent, by default it clears its data source positions. You can specify that it retain its positions for future calls by using the KEEP option, as described in *Optimizing Performance: Data Continuity and Memory Management* on page 4-20.

The Advantages of Modularizing Source Code Using CALL

Modularizing source code into several procedures has many advantages. One benefit is that you can use multiple procedures—executed via the CALL command—to share common source code among many developers, speeding up both development and maintenance time. For example, a generalized error message display procedure could be used by all WebFOCUS Maintain developers. After passing a message to the generalized procedure, the procedure would handle message display. The developers do not need to worry about how to display the message, and the error messages will always look consistent to end users.

Another advantage of modular design is that you can remove infrequently-executed source code from a procedure and move it into its own procedure. This reduces the size of the original procedure, simplifying its logic, making maintenance easier, and using less memory if the new procedure isn't called.

Optimizing Performance: Data Continuity and Memory Management

By default, when you terminate a child procedure, Maintain clears its data from memory to save space. You can optimize your application's performance by specifying, each time you terminate a child procedure, how you want Maintain to handle the procedure's data. You have two options, based on how often you will call a given procedure over the course of an application. If you will call the procedure:

- **Frequently**, use the KEEP option to make the procedure run faster by retaining its data between calls.

This option provides data continuity; the procedure's data carries over from the end of one invocation to the beginning of the next. That is, the next time you call the procedure, its variables and data source position pointers start out with the same values that they held when the procedure was last terminated. You can use these values or reinitialize them using the DECLARE (or COMPUTE) and REPOSITION commands.

Of course, variables passed by the parent procedure are not affected by data continuity since the child procedure receives them directly from the parent procedure at the beginning of each call.

KEEP's effect on transaction integrity. The KEEP option does not issue an implied COMMIT command at the end of a child procedure. When a child procedure with an open logical transaction returns to its parent procedure and specifies KEEP, the transaction continues into the parent.

- **Rarely**, use the RESET option to reduce memory consumption by freeing the procedure's data at the end of each call.

This option does not provide data continuity; all of the procedure's variables and data source position pointers are automatically initialized at the beginning of each procedure.

RESET's effect on transaction integrity. The RESET option issues an implied COMMIT command at the end of a child procedure. When a child procedure with an open logical transaction returns to its parent procedure using RESET, the transaction is closed at the end of the child procedure.

For more information about transactions spanning procedures, see *Ensuring Transaction Integrity* in *Using WebFOCUS Maintain*.

You can specify how a procedure will handle its data in memory by terminating it with the GOTO END command qualified with the appropriate memory-management phrase. The syntax is

```
GOTO END [KEEP|RESET];
```

where:

KEEP

Terminates the procedure, but keeps its data—the values of its variables and data source position pointers—in memory. It remains in memory through the next invocation, or (if it is not called again) until the application terminates. The procedure does not issue an implied COMMIT command to close an open logical transaction.

RESET

Terminates the procedure, clears its data from memory, and issues an implied COMMIT command to close an open logical transaction. This is the default.

You can use both options in the same procedure. For example, when you are ready to end a child procedure, you could evaluate what logic the procedure will need to perform when it is next called and then branch accordingly either to keep data in memory, saving time and providing data continuity, or else to clear data from memory to conserve space.

Executing External Procedures

You can execute external procedures—that is, procedures containing logic other than Maintain language commands—using the EXEC command. An external procedure is a FOCEXEC (also known as an iWay stored procedure).

You can use an external procedure to embed existing 3GL or DBMS stored procedure logic in a WebFOCUS Maintain application, to implement logic using static (compiled) SQL, and to update legacy systems via a transaction monitor program such as CICS or IMS/TM.

Within an external procedure you can include iWay API function calls, FOCUS Dialogue Manager commands, and commands written in the FOCUS command language (such as a WebFOCUS report procedure). You can execute many other kinds of logic from an external procedure, including:

- A compiled program, written in a language such as C or COBOL, that can be called on an iWay Data Server.
- A transaction running under the control of a transaction processing monitor such as CICS or IMS/TM.
- An executable file of commands written in a proprietary RDBMS language.

You can execute these other kinds of logic by issuing API function calls—such as EDARPC, CALLPGM, CALLORA, CALLSYB, and CALLIMS—in your external procedure.

For more information about FOCEXECs (also called procedures) and Dialogue Manager, see *Creating Reports With WebFOCUS Language* in your WebFOCUS Developer Studio documentation.

For more information about iWay stored procedures, please see the iWay documentation. For information about using API function calls, see the iWay documentation.

You can execute an external procedure from a Maintain procedure in the same manner in which you execute one Maintain procedure from another: simply substitute EXEC for CALL.

You identify the location of the child procedure in a deployment scenario in the Maintain Development Environment. At deployment time, the Maintain Development Environment automatically informs the parent procedure of the location of the child procedure by supplying the AT *server* phrase in the EXEC command. Do not code the AT *server* phrase yourself.

Example Executing an External Procedure

Consider the following EmpList procedure:

```
MAINTAIN FILE Employee
.
.
.
EXEC NewClass INTO EmpStk;
.
.
.
END
```

This calls the NewClass iWay stored procedure on the EducServ WebFOCUS Server:

```
TABLE FILE EMPLOYEE  
  PRINT LAST_NAME CLASS  
  ON TABLE PCHOLD  
END
```

In this example, the answer set returned by the NewClass iWay stored procedure populates the EmpStk stack in the Maintain client procedure.

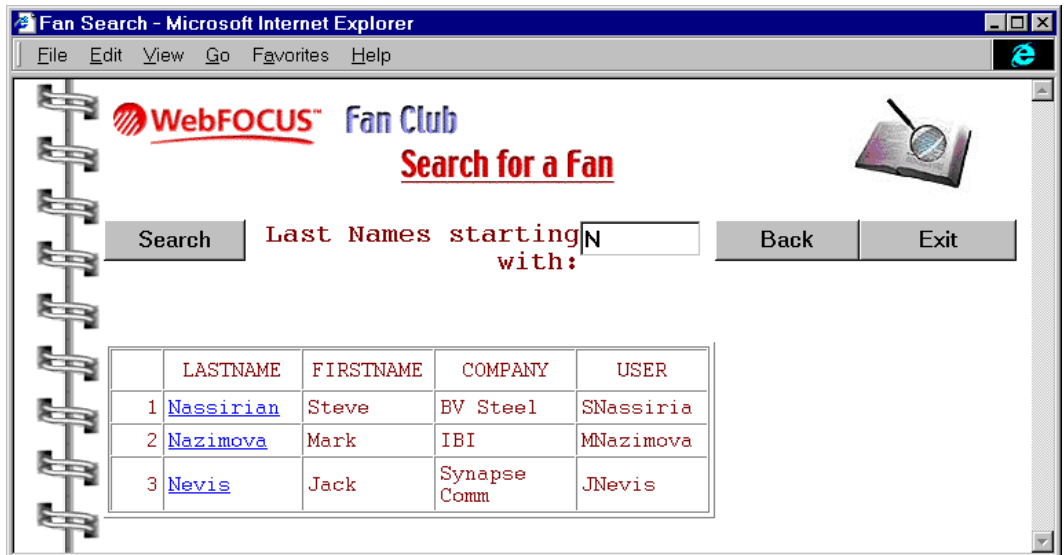
Forms and Event-driven Processing

Forms are the visual interface to a WebFOCUS Maintain application, giving it a dynamic and attractive face while enabling you to make the application flexible and to place its power at the fingertips of the application's end users.

WebFOCUS Maintain supports a full set of controls, such as list boxes, buttons, and ActiveX components, by which end users can manipulate data and drive the application. You can design forms that enable end users to:

- Enter and edit data.
- Select options.
- Perform business logic, such as searching a data source for a customer order.
- Send e-mail.
- Navigate the World Wide Web.
- Read application-specific help information.
- Control the flow of an application using an event-driven paradigm.

You develop these forms and the associated logic using the Form Editor. This is a sample form:



The screenshot shows a web browser window titled 'Fan Search - Microsoft Internet Explorer'. The browser's address bar is empty. The page content includes the 'WebFOCUS™ Fan Club' logo, a search bar with the text 'Search for a Fan', and a search button. Below the search bar, there is a text input field with the value 'N' and a label 'Last Names starting with:'. To the right of the input field are 'Back' and 'Exit' buttons. Below the search bar, there is a table with the following data:

	LASTNAME	FIRSTNAME	COMPANY	USER
1	Nassirian	Steve	BV Steel	SNassiria
2	Nazimova	Mark	IBI	MNazimova
3	Nevis	Jack	Synapse Comm	JNevis

Forms are event-driven, and enable:

- **Event-driven processing.** Forms are responsive to the needs of a user because they recognize user activity on the screen—that is, different types of screen events. For example, a form recognizes what the user does on the screen with the keyboard and mouse; it knows when a user clicks a button or changes a field value.

Forms also enable you to assign event handlers to these events; each time a specified event occurs, Maintain automatically triggers the corresponding event handler. If you use events to trigger the application's business logic, you can give the user more freedom—for example, over which editing tasks to perform, and in which order. You can also give the user access to more functionality, and more types of data, on a single screen. Event-driven processing gives the user more flexibility over the application, even as it gives the application more control over the user interface.

- **Event-driven development.** WebFOCUS Maintain provides you with a simple way of developing event-driven applications: event-driven development. Because much of an application's flow can be controlled from forms, you can develop the application as you paint its forms; first design the visual layout, then create controls, and finally code event handlers for the controls, all from the Form Editor. WebFOCUS Maintain also offers the Language Wizard which generates code for you, making it faster and easier to develop effective interfaces and powerful applications.

For an introduction to using forms and developing event-driven applications, see *How to Use Forms* on page 4-25, *Designing Event-driven Applications* on page 4-26, and *Creating Event-driven Applications* on page 4-26. Forms, and the Form Editor used to design and create them, are described in greater detail in *Developing and Using Forms* in *Developing WebFOCUS Maintain Applications*.

How to Use Forms

Forms are deployment-independent. You design a form to meet the needs of your application; WebFOCUS Maintain automatically implements the form as a Web page. This enables you to focus on logic, and leave implementation details to WebFOCUS Maintain.

Forms have standard form features, including:

- A title bar that identifies the form.
- Scroll bars that enable you to move a control's contents vertically and horizontally if they extend beyond the control's border.

In a Web-deployed application, forms are displayed one at a time in one Web browser session.

You can transfer control from one form to another; from a form to another Maintain procedure or to an external procedure; and from a form to any Internet resource, such as an e-mail client, a Web page, or an FTP server.

Designing a Form

Forms offer a diverse set of ways by which an application end user can select options, invoke procedures, display and edit fields, and get helpful information. For example, if you want the user to select an option or procedure, you can use any of the following controls:

- **Buttons.** You can specify a function to be performed when the end user clicks a button or image. Common examples are Done and Cancel buttons.
- **Radio buttons.** The end user can select one of a mutually exclusive group of options. For example, an employee could identify his or her department.
- **Check boxes.** The end user can select or deselect a single option or characteristic. For example, an applicant could indicate if this is the first time that he or she has applied.
- **Combo boxes and list boxes.** The end user can select one or more options from a dynamic list of choices.
- **Menus.** The end user can select an option from a drop down menu or submenu.

If you want to display or edit data, you can use these controls:

- **Edit boxes** to edit a single value.
- **Multi-line edit boxes** to edit a long value wrapping onto multiple lines.

- **HTML tables** to view a data source stack.
- **Grids** to edit and view a data source stack.

Designing Event-driven Applications

The flow of control in conventional processing is mostly pre-determined—that is, the programmer determines the few paths that the user will be able to take through the procedure.

To make your application's user interface more responsive to the user, WebFOCUS Maintain offers event-driven processing. Each time that an event occurs, it invokes, or *triggers*, the assigned event handler. In WebFOCUS Maintain, the event is something the application end user does in a form, and the event handler is a function or a URL. For example, you might create a button that, when clicked by a user, triggers an event handler that reads a data source and displays the data in the form.

Creating Event-driven Applications

Developing a procedure by writing out sequential lines of source code may be sufficient for conventional linear processing, but event-driven processing demands event-driven development. Developing an application in this way lets you build much of the application's logic around the user interface. In effect, you develop the application as you develop the interface in the Form Editor. For example, you could start by creating a form, creating a control, and then coding an event handler for one of the control's events. WebFOCUS Maintain also provides you with a number of automated tools for developing applications. For example, you can use the Language Wizard to generate source code for various operations such as retrieving and updating data.

Reading From a Data Source

Most applications need to read data from a data source. The most common method is to read data from a data source into a data source stack. Before reading, you first need to select the record in which the data resides. There are five ways of selecting records:

- **By field value for an entire set of records.** Use the NEXT command. The WHERE phrase enables you to select by value, and the FOR ALL phrase selects the entire set of records that satisfy the WHERE selection condition. The basic syntax for this is:

```
FOR ALL NEXT fields INTO stack WHERE selection_condition;
```

- **By field value for a sequence (subset) of records.** Use the NEXT command. This is similar to the technique for a set of records, except that it employs the FOR *n* phrase, selecting—at the current position in the data source—the first *n* records that satisfy the WHERE condition. The basic syntax for this is:

```
FOR n NEXT fields INTO stack WHERE selection_condition;
```


- **By field value, one segment at a time, one record at a time.** Use the MATCH command. The basic syntax for this is:
`MATCH fields [FROM stack] [INTO stack];`
- **Sequentially for a sequence (subset) of records.** Use the NEXT command. This technique employs the FOR *n* phrase to select the next *n* records. The basic syntax for this is:
`FOR n NEXT fields INTO stack;`
- **Sequentially, one segment instance at a time, one record at a time.** Use the NEXT command. The basic syntax for this is:
`NEXT fields [INTO stack];`

You can read from individual data sources, and from those that have been joined. Maintain supports joins that are defined in a Master File. For information about defining joins in a Master File, see *Describing Data With WebFOCUS Language*. (Maintain can read from joined data sources, but cannot write to them.)

You can evaluate the success of a command that reads from a data source by testing the FocError system variable, as described in *Evaluating the Success of a Simple Data Source Command* on page 4-28.

The NEXT and MATCH commands are described in detail in *Language Reference*.

Repositioning Your Location in a Data Source

Each time you issue a NEXT command, Maintain begins searching for records from the current position in the data source. For example, if your first data source operation retrieved a set of records

```
FOR ALL NEXT CustID INTO SmokeStack
  WHERE ProdName EQ 'VCR DUST COVER';
```

then Maintain will have searched sequentially through the entire data source, so the current position marker will now point to the end of the data source. If you then issue another NEXT command

```
FOR ALL NEXT LastName FirstName INTO CandyStack
  WHERE ProdName EQ 'CANDY';
```

Maintain searches from the current position to the end of the data source. Since the current position *is* the end of the data source, no records are retrieved.

When you want a NEXT command to search through the entire data source (as is often the case when you wish to retrieve a set of records) you should first issue the REPOSITION command to move the current position marker to the beginning of the data source.

Example Repositioning to the Beginning of the Data Source

The following REPOSITION command specifies the CustID field in the root segment, and so moves the current position marker for the root segment chain and all of its descendant chains back to the beginning of the chain (in effect, back to the beginning of the data source):

```
REPOSITION CustID;  
FOR ALL NEXT LastName FirstName INTO CandyStack  
WHERE ProdName EQ 'CANDY';
```

Writing to a Data Source

Writing to a data source is the heart of transaction processing applications. Maintain provides the following commands to write to a data source:

- **INCLUDE**, which adds the specified new segment instances to a data source.
- **UPDATE**, which updates the specified fields in a data source.
- **REVISE**, which adds new segment instances and updates the specified fields in existing segment instances.
- **DELETE**, which removes the specified segment instances from a data source.

These commands are described in detail in *Language Reference*.

Maintain requires that data sources to which it writes have unique keys.

Evaluating the Success of a Simple Data Source Command

When you issue a command that reads or writes to a data source, you should determine if the command was successful. Reasons for a data source command not being successful include attempting to insert a record that already exists, to update a record that does not exist, to delete a record that does not exist, to read a record that does not exist, and being interrupted by a system failure.

When you issue a command that reads or writes to a data source, if the command is:

- **Successful**, Maintain automatically sets the transaction variable FocError to 0 (zero), and writes to the data source.
- **Unsuccessful**, Maintain sets FocError to a non-zero value, and does not write to the data source.

Example Evaluating the Success of an UPDATE Command

The following function updates the VideoTrk data source for new video rentals. If the UPDATE command is unsuccessful, the application invokes a function that displays a message to the user. The line that evaluates the success of the command is shown in bold:

```
CASE UpdateCustOrder
    UPDATE ReturnDate Fee FROM RentalStack;
    IF FocError NE 0 THEN PERFORM ErrorMessage;
ENDCASE
```

Evaluating the Success of a Stack-based Write Command

When you write a set of stack rows to a data source, if you specify more rows than there are matching data source records, this does not invalidate the write operation; Maintain attempts to write all the matching rows to the data source. For example, the following UPDATE command specifies 15 rows, but there are only 12 matching rows. All 12 are written to the data source.

```
FOR 15 UPDATE Curr_Sal FROM NewSalaries;
```

When you write a set of stack rows to a data source, if one row fails, the following happens:

- The rows preceding the failed row are written to the data source.
- The rows following the failed row are ignored.
- FocError is set to a non-zero value, signaling an error.
- FocErrorRow is set to the number of the failed row.

Data source logic errors include attempting to insert an existing record, to update a nonexistent record, and to delete a nonexistent record.

To determine if an entire stack was successfully written to the data source, test FocError immediately following the data source command. If FocError is not 0, you can determine which row caused the problem by testing FocErrorRow; you can then reprocess that row. (If you will be passing control to a different procedure and reprocessing the row there, consider first setting the stack's FocIndex variable to the value of FocErrorRow in the current procedure, so that after you pass control the stack is already positioned at the problem row.)

If you do not wish to take advantage of this flexibility, and instead prefer to invalidate *all* the rows of the stack if any of them are unsuccessful, you can bracket the data source command in a logical transaction that you can then roll back. Logical transactions are discussed in *Transaction Processing* on page 4-30.

Row failure when committing to a data source. If a stack-based write command is part of a logical transaction, and the write command succeeds when it is issued but fails *when the application tries to commit the transaction*, Maintain of course rolls back all of the write command's rows, along with the rest of the transaction. (For example, a write command might fail at commit time because another user has already changed one of the records to which the command is writing.) Transaction processing is described in *Transaction Processing* on page 4-30.

Example Evaluating a Stack-based Update Command

The NewSalaries stack has 45 rows. The following command updates the Employee data source for all the rows in NewSalaries:

```
FOR ALL UPDATE Curr_Sal FROM NewSalaries;
```

If there is no data source record that matches the thirtieth row of NewSalaries, Maintain updates the data source records matching the first 29 rows and ignores records that match rows 30 and higher.

Transaction Processing

You are familiar with individual data source operations that insert, update, or delete data source segment instances. However, most applications are concerned with “real-world” transactions, such as transferring funds or fulfilling a sales order, that each require several data source operations. These data source operations may access several data sources, and may be issued from several procedures. We call such a collection of data source operations a *logical transaction*. (It is also known as a logical unit of work.)

The advantage of describing a group of related data source commands as one logical transaction is that the transaction is written to the data source only if all of its component commands are successfully written to the data source. Transaction integrity is an all-or-nothing proposition: if even one part of the transaction fails when you try to write it (by issuing the COMMIT command), Maintain automatically rolls back the entire transaction, leaving the data source unchanged.

Transaction integrity also ensures that when several users share access to the same data source, concurrent transactions execute as if they were isolated from each other; a transaction's changes to a data source are concealed from all other transactions *until that transaction is committed*. This prevents each transaction from being exposed to interim inconsistent images of the data source, and so protects the data from corruption.

There are many strategies for managing concurrent data source access. No matter which type of data source you use, Maintain respects your DBMS's concurrency strategy and lets it coordinate access to its own data sources.

Transaction processing is described in greater detail in *Ensuring Transaction Integrity* in *Developing WebFOCUS Maintain Applications*.

Example Logical Transactions in a Bank

A banking application would define a transfer of funds from a checking account to a savings account as one logical transaction comprising two update operations:

- Subtracting the funds from the source account (`UPDATE Checking FROM SourceAccts`).
- Adding the funds to the target account (`UPDATE Savings FROM TargetAccts`).

Why? If the application had not been able to subtract the funds from the checking account, because someone had cleared a check against that account a few moments earlier and depleted its funds, *but* the application had added the funds to the savings account, the bank's accounts would become unbalanced.

The two update commands (subtracting and adding funds) must be described as parts of a single logical transaction, so that the subtraction and addition updates are not written to the data source independently of each other.

Classes and Objects

Most application development is modular: the developer creates complex systems comprised of smaller parts. In “conventional” development, these modules are processes (such as procedures). In object-oriented development, the modules are models of real-world objects (such as a customer or a shipping order). Each object encapsulates both data and processes.

For example, if you are developing an order fulfillment system for a mail-order clothing business, the objects might include customers, orders, and stock items. A customer object's data might include the customer's ID code, phone number, and order history; the customer's processes might include a function that adds the customer to a new mailing list, a function that updates the customer's contact information, and a function that places an order for the customer.

Object-oriented development—because it models the real-world objects with which your enterprise deals, and encourages you to reuse application logic in a variety of ways—is a more efficient way of developing applications. WebFOCUS Maintain enables you to create applications using object-oriented development, conventional development, or a hybrid of these two methods, providing you with a flexible path.

What Are Classes and Objects?

Most applications need many objects of the same type. For example, if your business has 500 customers, you need one object to represent each customer. No one would want to design a customer object 500 times; clearly, you need a template that defines all customer objects, so that you can design the template once, and use it often—each time you create a new customer object to represent a new customer.

An object's template is called its *class*. Each object is an instance of a class. The class defines what type of object it is—in fact, when you create a class, it becomes a new data type, one which you can use to define an object, in the same way that you can use a built-in data type like integer or alphanumeric to define a simple variable like a customer code.

You define a class by describing its properties. Classes have two kinds of properties:

- **Data**, in the form of the class's variables. Because these variables exist only as members of the class, they are called *member variables*. (In some object-oriented development environments these are also known as object attributes or instance variables.)
- **Processes**, implemented as functions. Because these functions exist only as members of the class, they are called *member functions*. (In some object-oriented development environments these are also known as methods.)

If you want to create a new class that is a special case of an existing class, you could derive it from that existing class. For example, in a human resources application, a class called Manager could be considered a special case of a more general class called Employee: all managers are employees, and possess all employee attributes, plus some additional attributes unique to managers. The Manager class is derived from the Employee class, so Manager is a subclass of Employee, and Employee is the superclass of Manager.

A subclass inherits all of its superclass's properties (that is, it inherits all of the superclass's member variables and member functions). When you define a subclass you can choose to override some of the inherited member functions, meaning that you can recode them to suit the ways in which the subclass differs from the superclass. You can also add new member functions and member variables that are unique to the subclass.

Index

Symbols

\$\$Declarations comment 2-29

Numerics

3GL programs 1-2

A

addafan.gif 2-2

aligning controls 2-18

anchor controls 2-18

applications 1-2, 2-2, 4-23
 application logic 1-3
 closing at run time 2-44–2-45
 developing 1-4
 editing 2-6
 event-driven 4-23, 4-26
 event-driven processing 4-26
 modularizing 4-20
 running locally 2-21

B

Back button 2-44

background images 2-46, 2-51
 tiling 2-50

BackgroundImage property 2-46, 2-51

Binding the Selection Result dialog box 2-25

button control 2-27
 adding to forms 2-27

C

C/C++ programs 1-2

CALL command 4-15–4-16
 example 4-19
 modularizing code 4-20
 passing variables 4-17

CASE command 2-29

child procedures 4-15
 accessing data sources in 4-19

CICS transactions 1-2

classes 4-31

Click event 2-35

COBOL programs 1-2

columns in data source stacks 2-12, 4-8–4-9

comments 2-29

COMMIT command 4-19

COMPUTE command 4-8

concurrent processing 4-30

Control Columns dialog box 2-39

controls
 aligning 2-18
 grouping 2-16
 moving 2-15
 selecting multiple 2-18

Controls palette 2-14

Create a Project dialog boxes 2-2

Create new project file dialog box 2-4

cross-referenced data sources 4-26

Current Area columns 4-13

currfan.gif 2-2

- Customer Support Service 1-v
 - How to contact 1-v
 - Information required 1-v

D

- data continuity 4-20
- data source descriptions 2-7
- data source stacks 2-12, 4-2
 - clearing 2-34–2-35
 - columns 4-5–4-6
 - compared to SPA 4-2
 - copying data 4-9–4-10
 - creating 2-13, 4-4–4-5
 - Current Area 4-13
 - data source-derived columns 4-5
 - editing 4-12
 - extracting data from data sources 2-38
 - implied columns 2-26
 - looping through 4-11–4-12
 - optimizing performance 4-14
 - rows 4-10
 - sorting 4-12
 - user-defined columns 4-8
 - writing to data sources 2-29, 2-31
- data sources
 - accessing in child procedures 4-19
 - adding fields to forms 2-11–2-13
 - adding to projects 2-7–2-8
 - concurrent transactions 4-30
 - displaying data from 2-39
 - expanding in Project Explorer 2-9
 - extracting data 2-38
 - joined data sources 4-5
 - keys 4-28
 - logical transactions 4-30
 - position in 4-19, 4-27
 - reading 4-26, 4-28
 - sharing 4-30
 - updating records 2-31
 - using in procedure 2-9
 - writing to 2-27, 4-28–4-29

- Declarations comment 2-29
- DEFINE attribute 4-8
- DELETE command 4-28
- deploying projects 2-21
- Display all files in the project path button 2-7

E

- embedded joins 4-5
 - reading 4-26
- END command 2-29
- ENDCASE keyword 2-29
- Enter a List Item dialog box 2-22
- Event Handler editor 2-35
- event-driven processing 4-23, 4-26
- events 2-35, 4-23
- EXEC command 4-21
- Exit button 2-44–2-45
- Explorer 2-6
 - using folders 2-50
- Explorer toolbar
 - Display all files in the project path 2-7
- Explorer, Project 2-8, 2-14
 - expanding data sources 2-9
- external procedures 4-21
 - rules for using the AT phrase 4-21

F

- Fan Club application 2-1
 - creating forms 2-11
 - displaying fans 2-37
 - requirements 2-2
 - running locally 2-21
 - saving 2-20
 - writing to a data source 2-27

fan.gif 2-2

FANNAMES data source 2-2
 adding fields to forms 2-11
 displaying data from 2-39
 using in procedures 2-29

fields 2-9, 4-8
 adding to forms 2-11–2-13
 Current Area and 4-13
 renaming prompts 2-16, 2-18

flow of control 4-15
 looping 4-11–4-12

FocCount variable 4-10

FocErrorRow variable 4-29

FocIndex variable 4-10

FOCUS code 1-2

folders in the Explorer 2-50

FOR keyword 4-4

Form Editor 2-14, 4-26
 displaying rulers 2-52

forms 4-23, 4-25
 closing at run time 2-44
 creating 2-37–2-38, 4-25
 defining stack columns 4-5
 event-driven processing 4-26
 events 4-23
 invoking procedures 4-25
 linking 2-42
 naming 2-26
 selecting options 4-25

functions
 assigning to events 2-35
 creating 2-28
 editing 2-29
 viewing source code 2-29

G

GOTO command 4-20

graphics 2-46

grid in Form Editor 2-15

H

hierarchical data sources 2-9

HTML Table control 2-39
 viewing scroll bars 2-44

I

Image Source dialog box 2-46, 2-51

images 2-46, 2-52
 tiling 2-50

implied columns 2-26

IMS/TM transactions 1-2

INCLUDE command 2-31, 4-28

INFER command 2-29

K

KEEP keyword 4-20

key fields 2-9, 4-28
 requirements 4-28

L

Language Wizard
 adding records to a data source 2-31
 clearing data source stacks 2-34–2-35
 extracting data from data sources 2-38

List Source dialog box 2-22

logical transactions 4-30
 concurrent transactions 4-30

M

- MAINTAIN command 2-29
- Maintain Controls palette 2-14
- Maintain language 1-2
- Maintain Language Wizard
 - adding records to a data source 2-31
 - clearing data source stacks 2-34–2-35
 - extracting data from data sources 2-38
- Maintain procedures
 - accessing data sources 4-19
 - child 4-15
 - components 2-14
 - creating 2-4
 - data source position 4-19
 - event driven 4-23
 - flow of control 4-15
 - passing variables 4-17, 4-19
 - remote 4-15–4-17, 4-19
 - viewing source code 2-29
- MATCH command 4-26
 - defining stack columns 4-5
- memory management 4-20
- modular source code 4-20

N

- New Function dialog box 2-28
- New Virtual Folder dialog box 2-50
- NEXT command 2-38, 4-5
 - defining stack columns 4-5
 - setbased processing 4-26
- n-tier applications 1-3

O

- objects 4-31
- operating systems 1-2

- Overflow property 2-44

P

- performance 4-20
 - memory management 4-20
- pictures 2-46
- platforms 1-2
- procedures, Maintain
 - accessing data sources 4-19
 - child 4-15
 - components 2-14
 - creating 2-4
 - data source position 4-19
 - event driven 4-23
 - flow of control 4-15
 - passing variables 4-17, 4-19
 - remote 4-15–4-17, 4-19
 - viewing source code 2-29
- procedures, WebFOCUS 1-2
- Project Explorer 2-8, 2-14
 - expanding data sources 2-9
- projects 2-2
 - creating 2-2
 - running 2-21
 - running locally 2-21
 - saving 2-20

- prompts for fields 2-16, 2-18

- property sheet 2-14, 2-16

R

- Radio button control 2-22
- RDBMS stored procedures 1-2
- records 4-2
 - adding 2-31
 - deleting 4-2
 - selecting 4-2
 - updating 4-2

- relational data sources 2-9
- remote procedures 4-15
- REPEAT command 4-11–4-12
- REPOSITION command 2-38, 4-19, 4-27
- RESET keyword 4-20
- Resource Wizard 2-46
- rulers 2-52

S

- Save button/command 2-20
- saving your work 2-20
- Scratch Pad Area (SPA) 4-2
- scroll bars for HTML tables 2-44
- segments 2-9
- Select Segment Fields dialog box 2-12–2-13
- SelectedItem property 2-25
- servers 1-2–1-3
- set-based processing 4-2, 4-26
 - methods 4-4
- Show All Files command 2-7
- SHOW keyword in WINFORM command 2-29
- source code for procedures 2-29
- SPA (Scratch Pad Area) 4-2
- spiralbg.gif 2-2
- stacks 2-12, 4-2
 - clearing 2-34–2-35
 - columns 4-5–4-6
 - compared to SPA 4-2
 - copying data 4-9–4-10
 - creating 2-13, 4-4–4-5
 - Current Area 4-13
 - data source-derived columns 4-5

- stacks (*continued*)
 - editing 4-12
 - extracting data from data sources 2-38
 - implied columns 2-26
 - looping through 4-11–4-12
 - optimizing performance 4-14
 - rows 4-10
 - segments 4-5
 - user-defined columns 4-8
 - writing to data sources 2-29, 2-31

- stored procedures in Maintain applications 1-2
- syntax color 2-29

T

- Table Column dialog box 2-39, 2-41
- temporary fields 4-8
 - redefining in child procedures 4-17
- Text property 2-16
- Title property 2-26
- Toggle grid button 2-15
- Toggle rulers button 2-52
- ToolTipText property 2-24
- Top function 2-14, 2-29
- transaction processing 4-30
 - concurrent transactions 4-30
- transaction values 4-2
- triggers 4-23
- tutorials 2-1

U

- untitled forms 2-26
- UPDATE command 4-28
- Use these Data Sources in Procedure dialog box 2-9
- user-defined stack columns 4-8–4-9

V

- variables 4-20
 - memory management 4-20
 - passing between procedures 4-17, 4-19
- virtual fields 4-8
- virtual folders in the Explorer 2-50

W

- WebFOCUS Maintain 1-1–1-2, 1-4
 - benefits 1-4
 - developing applications 1-4
 - n-tier applications 1-3
- WebFOCUS procedures 1-2
- WebFOCUS Server 1-2–1-3
- WinClose function 2-44
- WinExit function 2-44
- WINFORM command 2-29

Reader Comments

In an ongoing effort to produce effective documentation, the Documentation Services staff at Information Builders welcomes any opinion you can offer regarding this manual.

Please use this form to relay suggestions for improving this publication or to alert us to corrections. Identify specific pages where applicable. You can contact us through the following methods:

Mail: Documentation Services - Customer Support
Information Builders, Inc.
Two Penn Plaza
New York, NY 10121-2898

Fax: (212) 967-0460

E-mail: books_info@ibi.com

Web form: <http://www.informationbuilders.com/bookstore/derf.html>

Name: _____

Company: _____

Address: _____

Telephone: _____ Date: _____

E-mail: _____

Comments:

Reader Comments